Applications of a Three-Dimensional Position and Attitude Sensing System for Neutral Buoyancy Space Simulation

by

Karl Gerard Kowalski

B.S., Massachusetts Institute of Technology (1986)

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

Master of Science
in
Aeronautics and Astronautics
at the
Massachusetts Institute of Technology
October, 1989

Signature of Author_____
Department of Aeronautics and Astronautics
October 27, 1989

Certified by_____
Prof. David Akin
Thesis Supervisor

Accepted by_____
Prof. Harold Y. Wachman
Chairman, Department Graduate Comittee

Applications of a Three-Dimensional Position and Attitude Sensing System for Neutral Buoyancy Space Simulation

by

Karl Gerard Kowalski

Submitted to the Department of Aeronautics and Astronautics
on October 27, 1989, in partial fulfillment of the requirements
for the degree of Master of Science in
Aeronautics and Astronautics

ABSTRACT

This thesis describes the design and applications of a three-dimensional position and attitude sensing system for use in support of neutrally-buoyant space telerobotic operations. The system makes use of acoustic pulses traveling underwater from predefined locations and picked up by hydrophones stationed on the vehicle whose position and attitude are to be determined. A surface computer retrieves the data acquired by the underwater system and converts the data into x, y, and z coordinates and the roll, pitch, and yaw attitudes of the test vehicle.

Both static and dynamic tests were conducted to determine the accuracy of the system. Trajectory experiments were conducted on two vehicles, the Beam Assembly Teleoperator (BAT) and the Multimode Proximity Operations Device (MPOD). Reach envelopes for humans both in and out of pressure suits were determined using the system. Conclusions and recommendations for future systems making use of this apparatus are included.

This work was conducted under NASA Grant NAGW-21.

Thesis Supervisor: David Akin

Acknowledgements

First and foremost, I would like to thank my undergraduate worker, Matthew Machlis, for all his wisdom and efforts in turning 3DAPS into a viable, worthwhile project of the SSL family. I kneel before thee, O Holy Machlis.

Next, of course, is Professor Dave Akin, without whose knowledge and support I would have given up long ago. John Spofford deserves recognition for the 3DAPS concept and prototype single-point 3DAPS. Also, thanks to Don Weiner, Earle Wassmouth, and Al Shaw, for all their help in creating much of the necessary hardware for my tests and for keeping the cookie tin filled most of the time. Thanks to Ping Lee, for letting me buy all kinds of neat stuff. Thanks to Max Ochoa for putting up with me and 3DAPS over the summer. Many thanks to the divers who put up with all the demands and paranoia of a landlubber like me: Russ Howard, Wisdom Coleman, Rich Patten, Terry Fong, Lisa Evelsizer, Paul Duncan, Joe DiMare, and Suzanne Garber. Thanks to Vicky Rowley for fixing 3DAPS for the last pool test, and for keeping me alive. Praise be to Ella Atkins for all those wonderful arguments and debates. I will be eternally grateful to Rob Sanner for all his help in advancing the 3DAPS state-of-the-art. And many thanks to the other members of SSL: Sayan, Wendy, Mary, Lisa R., Jud, Jim, Jennifer, Ender, Colleen, Beth, Andrew, and Alissa; all of whom provided wit if not always wisdom when 3DAPS was working and especially on those singular occasions when 3DAPS refused to speak to me.

I would also like to thank and acknowledge Janice Onanian, Nicholas Newell, and John "Dusty" Deyst for their support, humor, and advice.

Finally, this thesis is dedicated to my parents, in recognition of all their love and help and understanding while I was working and especially when I screwed up. Thanks, mom and dad!

All praise the Holy 3DAPS! Amen, Alleluia.

Table of Contents

List of Figures

1.0 Introduction

In 1978, the MIT Space Systems Laboratory (SSL) began to investigate the use of telerobotics in space operations. In order to greatly reduce human workload in space, the idea of teleoperated robots has been researched and more fully developed at the SSL. The objectives of this research were to determine the possible applications of humans and machines with regard to performing operations in space.

In support of the above objectives, the SSL has designed and constructed three prototype teleoperated robots. These robots were designed to operate underwater and are made neutrally buoyant with the aid of flotation elements. Operation of the robots underwater simulates the zero-gravity environment in which robots are expected to perform while in orbit, but does not require the robots to be transported there to be developed. Thus, experiments can be run at much lower cost than by sending the robots into space, and for far longer periods of time than the zero-g experiments performed in a KC-135 aircraft.

Through the development of these robots, the SSL has gained much knowledge and experience in the design and use of teleoperated vehicles, which can be utilized to improve the design of similar vehicles in the future. The SSL's primary focus has been to research space applications of the teleoperated robots, and consequently the design of telerobots to perform the requisite tasks. Such tasks include assembling and constructing space structures, and the transport and support of humans and materials in space.

The use of teleoperated robots in a neutrally buoyant environment has sparked collateral research into other areas, in order to support and improve the capabilities of the robots. One of the most important areas of research has been that of position determination, in an attempt to provide accurate location data to a feedback control system designed to regulate the motion of a robot. The drag induced by vehicular motion in a neutrally buoyant environment causes simulations performed in such

an environment to differ significantly from similar operations carried out in orbit by the same system. In order to more fully model operation in space, an accurate position and attitude determination system is required. In space, such a requirement could be fulfilled through the use of fixed radio beacons. However, in a neutrally buoyant environment such a setup will not work, due to the inability of radio waves to propagate through the medium. The use of accelerometers and gyroscopes can lead to erroneous position and attitude information due to the inaccuracies from integrating rates and accelerations to acquire displacements, and the use of pendulum inclinometers cannot completely determine the attitude of a vehicle. Therefore a new method of unambiguous position and attitude determination was developed. The Three-Dimensional Acoustic Positioning System (3DAPS) was the result of this research.

While radio waves cannot propagate significantly through a neutrally buoyant environment, acoustic waves are ideally suited for such a task. The 3DAPS system was devised to employ acoustic pulses to locate an underwater vehicle, specifically a neutrally buoyant space teleoperated robot. Unlike sonar, which uses reflections of transmitted acoustic pulses to determine range information, the acoustic pulses of the system are not intended to return from the vehicle whose position is to be determined. Instead, a counter on the vehicle is started when the acoustic pulse is transmitted. When the pulse is received, the counter is stopped. Because the counter is running at a known constant rate, and because the water in the neutral buoyancy facilities used is essentially isothermal, the count value when the pulse is received can be expressed as a range value between the pulse generator and the acoustic sensor (hydrophone) using the speed of sound. By using many pulse generators and knowing the locations of the pulse generators within a defined coordinate system, the location of the hydrophone can be determined within the same coordinate system. By utilizing at least three hydrophones fixed in a prescribed orientation on the vehicle, the unambiguous attitude of a vehicle carrying the hydrophones can be calculated

from their relative coordinates. By using the calculated coordinates and attitude as feedback data, the control of a neutrally buoyant teleoperated robot can be greatly enhanced.

Figure 1 shows an operational block diagram of the 3DAPS system as used for the objectives described in this thesis. The central block, the sequencer, controls the operation of the pulse generators ("thumpers") by commanding each to fire in turn, notifies the receiver box of thumper firings, and provides computer-receiver communications through fiber optic lines and an RS-232 connection. The receiver box performs the range-acquisition operations and passes range data to the monitoring computer, which processes the range data and calculates the coordinates of the hydrophones and the attitudes of the vehicle to which they are connected.

Questions arise as to the viability of the system. How accurate is the system? How well does it work with a vehicle in motion? How fast can the system operate? The purpose of this research was to first determine the capabilities of the system, and then demonstrate applications of the system in a neutrally buoyant environment. Also, experiments were designed to demonstrate the use of the system in other neutral buoyancy research projects. Two sets of performance tests were designed to determine the limitations of the system: these were the static and the controlled dynamic experiments. The static tests showed how accurate the system can be in an ideal situation. The controlled dynamic tests were designed to quantify the degradation in system accuracy due to motion of the vehicle to which it is attached.

The capabilities of the system were demonstrated by situating the receiver unit on a vehicle and allowing the vehicle to perform normal tasks. Also, the system was used to determine the reach envelope of humans, both in and out of pressure suits. This objective is a particularly useful application of the 3DAPS system as it allows a comparison to be made independent of NASA's method of determining human reach envelopes.

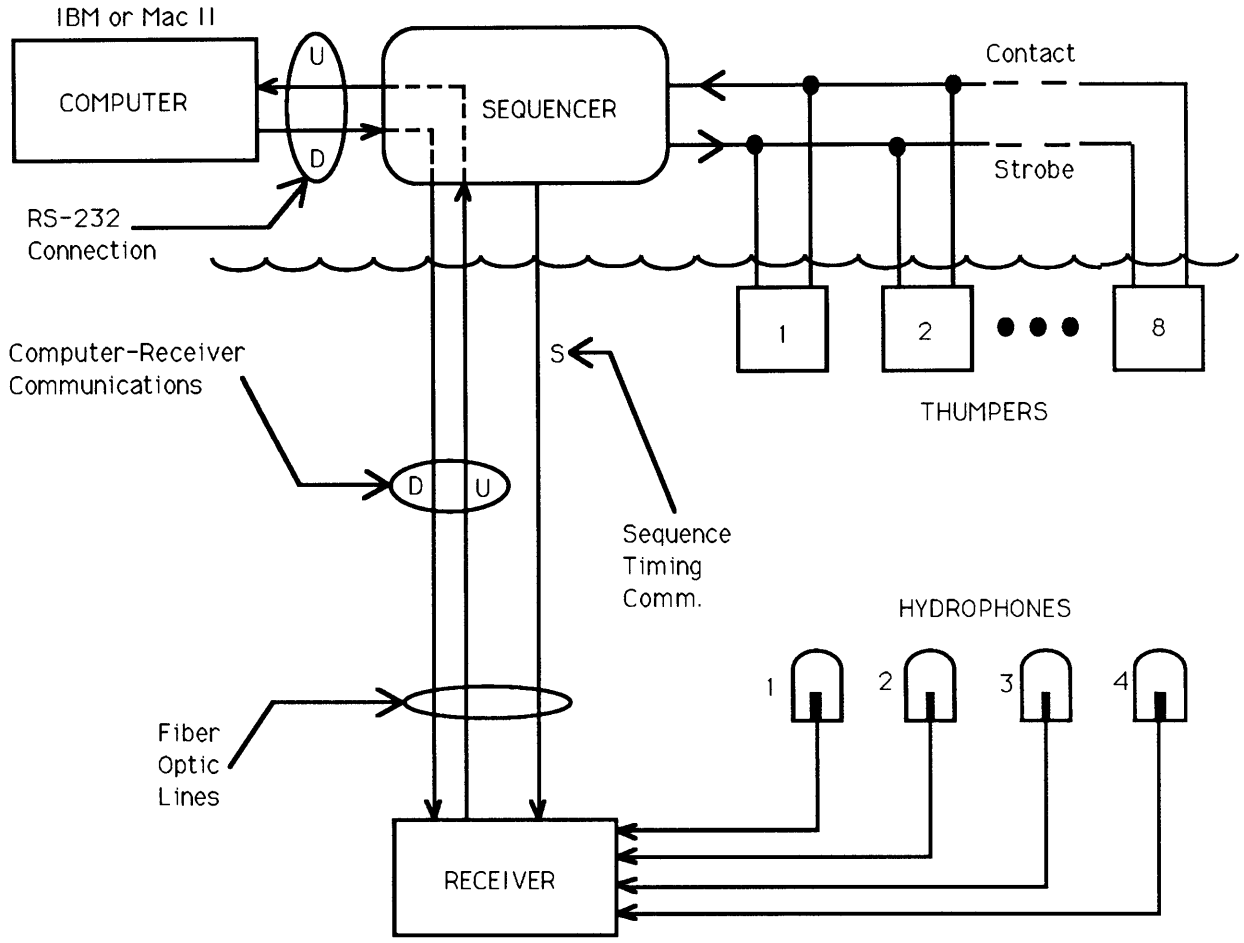


Figure 1. 3DAPS Operational Block Diagram

Finally, this thesis discusses possible future improvements in the design and performance of the 3DAPS system.

2.0 The 3DAPS System

In order to fully understand the objectives and experiments covered in this thesis, a knowledge of the system design is required. The system is based on an original SSL design documented in reference 1. While that system was built around only one hydrophone, the system documented here uses four hydrophones. However, the design of the present system very closely resembles that of ref. 1.

The system is best described by its major components. The diagram in Fig. 1 is most easily broken down into three categories: the sequencer and thumpers, the receiver and hydrophones, and the monitoring computer. As noted earlier, the purpose of the system is to provide position information through the use of acoustic pulses. The system operates as follows: the thumpers generate acoustic pulses, the sequencer coordinates the thumpers and tells the receiver when to start counting, the hydrophones listen for the acoustic pulses, the receiver determines the straight-line range between the hydrophones and the thumpers, and the monitoring computer calculates the coordinates of the hydrophones as determined by the ranges acquired.

The following sections will more fully describe the system components and their operation. All technical data — mechanical and electrical diagrams and specifications — can be found in reference 2.

2.1 The Sequencer and Thumpers

In order to provide a means for determining the distance between two points in a neutrally buoyant environment, an acoustic pulse generator (thumper) was designed and constructed at the SSL. A watertight container, consisting of a clear plastic tube and two black anodized aluminum endplates, houses the circuitry and mechanical hardware necessary to generate the acoustic pulses. To produce a pulse, an aluminum rod is accelerated by a solenoid and slammed into the aluminum endplate that points into the test area. The solenoid is powered by a 40 V DC source, and its operation

is controlled by a TTL circuit consisting mainly of a one-shot device (74LS123). The input to the one-shot is a strobe signal which produces an electronic pulse which causes the solenoid to activate and consequently generate the acoustic pulse. The thumpers were designed such that the pulse generated contains most of its energy near 100 kHz. When the solenoid is deactivated, a spring pulls the rod back from the endplate, readying it for its next cycle. While in contact with the endplate, the rod completes an electrical circuit which sends a TTL-level output signal, known as the contact signal, back to the sequencer.

To fully determine the x, y, and z coordinates of a hydrophone by using this system, a minimum of three range values must be obtained, from three independent sources. Thus, at least three thumpers must be employed to calculate coordinates based on range data. However, due to probable inaccuracies in the method of range determination and possible signal blockage, more thumpers are used than the minimum number necessary. The work performed in this thesis made use of a total of eight thumpers. Figure 2 shows schematically the relative positions of the thumpers when deployed in the MIT pool, while Fig. 3 shows the arrangement used at the neutral buoyancy facility at NASA's Marshall Space Flight Center. The arrangement of the thumpers at the MIT pool was such that the thumpers occupied the corners of a rectangular parallelepiped, while the Marshall tank configuration was approximately in the form of two rings, one high and one low, of four thumpers each. The positions of the thumpers were determined for tests at both facilities. At the MIT pool, the thumper positions were measured directly, with respect to a coordinate system with its origin at the surface in one corner of the pool, and with the z-axis pointing down. At the Marshall tank, the thumper coordinates could not be fully determined by direct measurement. Instead, straight-line ranges were measured from four known points in a coordinate system originating within the mockup of the shuttle bay at the bottom of the tank. The z-axis for the tank was defined as positive up, and the z coordinates for the thumpers were measured directly.

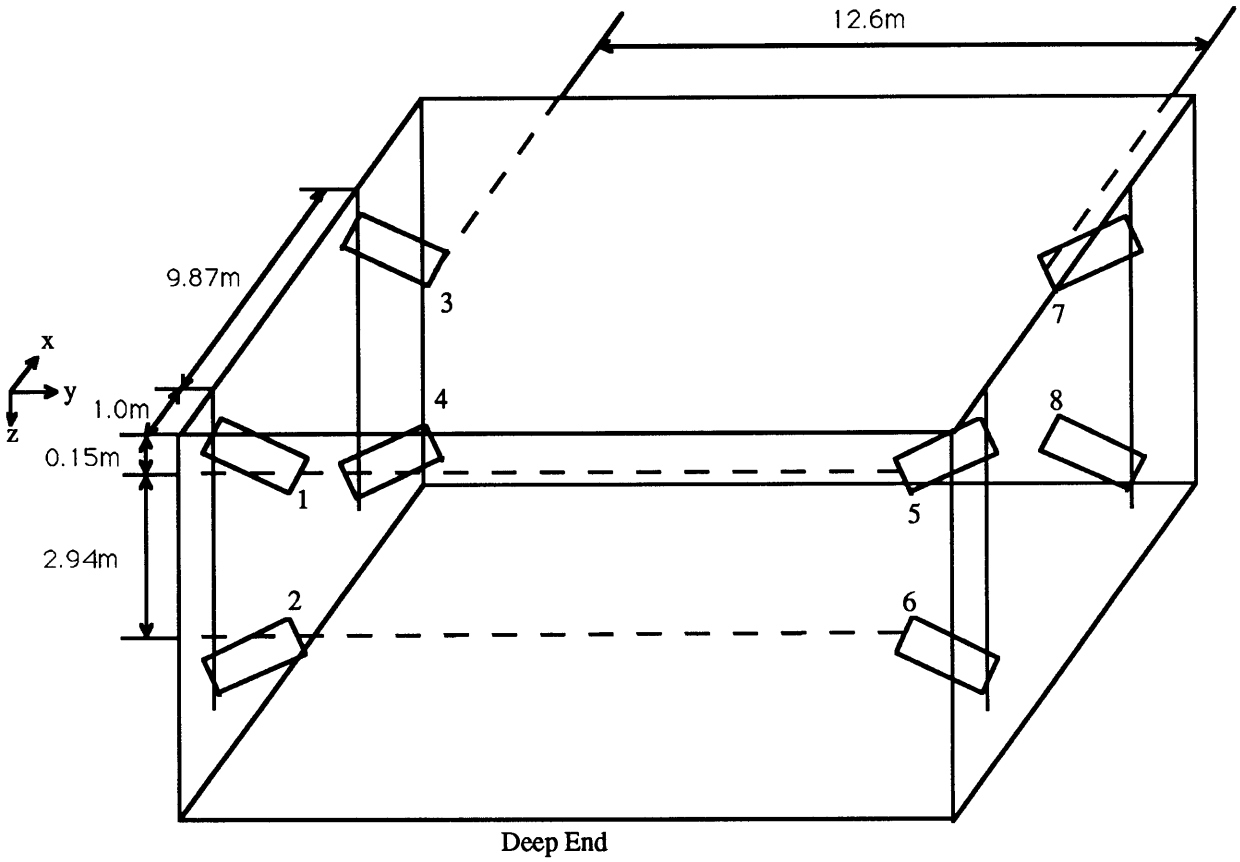


Figure 2. MIT Pool Thumper Locations

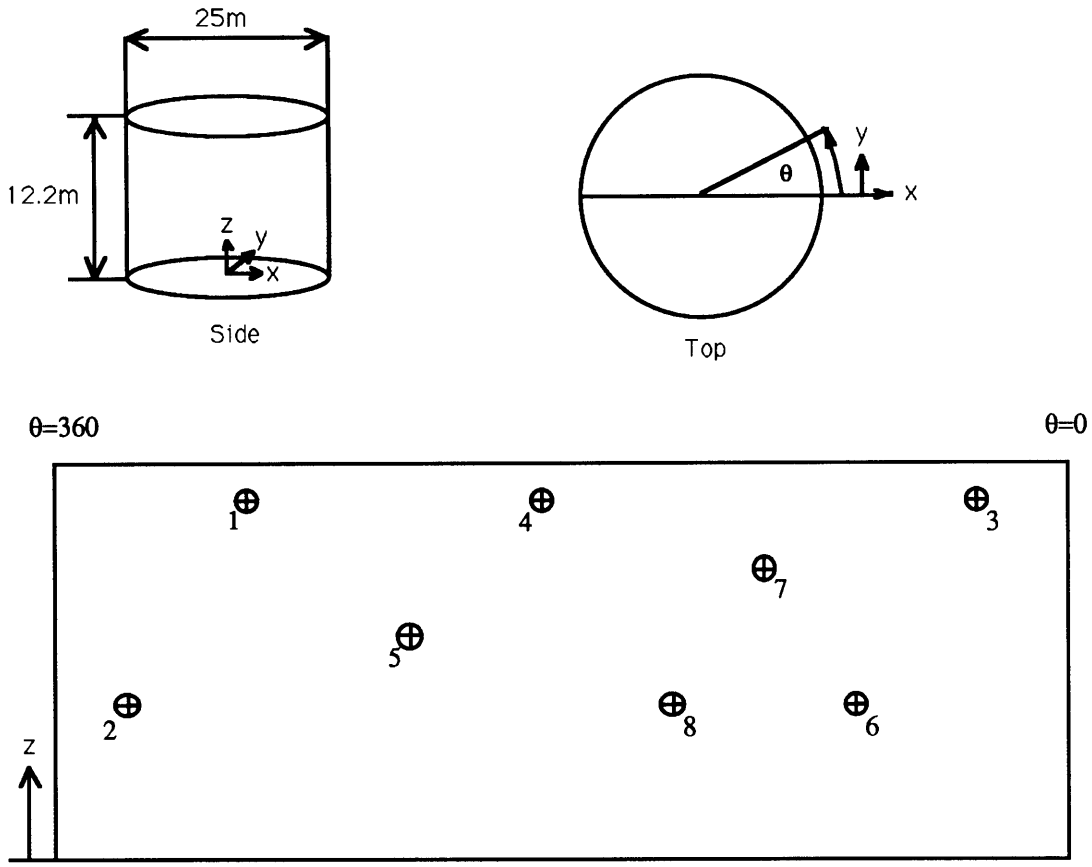


Figure 3. Marshall Tank Thumper Locations

The x and y coordinates of the eight thumpers were then determined by minimizing the error between the estimated ranges based on given thumper coordinates and the four known positions, and the actual ranges measured. Thus, the necessary fixed points of the system were determined. Appendix A contains the numerical data of the thumper positions at both the MIT pool and the Marshall tank.

Because thumper signals are indistinguishable, a means was necessary by which the receiver would be informed of which thumper had fired; thus, the receiver would be made aware of which range corresponds to which thumper. The sequencer was designed and constructed to achieve these requirements.

The reasoning behind the sequencer's design is as follows: up to eight thumpers are intended to be used to transmit acoustic pulses to the receiver. Each acoustic pulse is roughly identical to that from any other thumper, so the thumpers must be driven consecutively, in a sequence; hence the name, sequencer. Figure 4 shows the sequencer's front panel. An operator running an experiment with the system has control over several variables of sequencer operation. These include the number of thumpers in use, the time delay between each thumper firing, and the delay between the firing of the last thumper of one set and the first thumper of the next.

An operator can command the sequencer to use any number of thumpers from one to a maximum of eight. The number of thumpers used is needed for both the coordinate calculation routines in the monitoring computer and for the receiver range-acquisition software, as described below. One important requirement is that the last thumper of a sequence must always fire, otherwise the receiver software will be stopped waiting for it to fire. This is more fully discussed in the next section describing the receiver box.

The operator has control over the amount of time between thumper firings. The sequencer allows the user to vary the time delay from the firing of one thumper to the firing of the next, and this delay is set by a knob on the front panel of the sequencer. The reason for this delay is as follows. All the thumpers produce equivalent pulses,

necessitating that the thumpers be fired in a sequence, instead of firing all of them at the same time. The receiver is informed of a thumper firing (see below) and waits to receive the acoustic signal through the hydrophones. Some time must be allotted to the receiver between the firing of one thumper and the firing of the next so that the receiver can store the data from the first thumper and prepare for the next. Thus, as will be seen in the next section, the minimum possible delay would be the time needed by the receiver to count from zero to its maximum count value, which is about 16.5 milliseconds. However, in practice the thumper firing delay was determined empirically, by varying the delay while the system was in operation. The limiting value was determined as the smallest delay that allowed the receiver to fully acquire range data; if the delay was too short, the receiver would return zeros. The smallest average thumper firing delay was 0.22 seconds, and this is what was used for the experiments.

The user also controls a time delay between thumper sequences. This is set by a pushbutton on the sequencer which controls the number by which the firing delay is multiplied to arrive at the selected sequence delay. This time delay is necessary to permit the receiver to convert all the count values it has acquired into range values, transmit these numbers to the surface monitoring computer, allow the computer to calculate the coordinates, and get the new coordinates from the computer. The setting selected for this delay was chosen empirically, determined from trial and error by allowing the computer just enough time to determine the coordinates and send them to the receiver, without missing any sequence of pulses. The time delay used was 0.66 seconds, corresponding to a selected multiplying factor of two. This brought the total thumper sequence period to 2.2 seconds, which gives an information update rate of approximately 0.5 Hz.

Once the operator starts the sequencer, the sequencer functions automatically, independent of either the receiver box or the monitoring computer. This independence allows more than one 3DAPS receiver system to be in operation at any one time,

and the sequencer is at present capable of supporting communications for two sets of receivers and monitoring computers. As each selected thumper fires in sequence, a four-bit signal is sent through a high-speed fiber optic line to every receiver, representing the thumper's identification number. This signal is transmitted at a rate of 250 kilobaud; the fiber optic line and transmitter-receiver pair used for this signal are capable of handling up to one megabaud. After the thumper has fired, and assuming a valid contact signal has been received by the sequencer, the sequencer transmits a one-bit timing signal through the high-speed fiber optic line to indicate that the thumper has in fact fired. If the thumper does not fire or doesn't send a valid contact signal to the sequencer, no contact signal is sent to the receiver. The contact signal starts the receiver's counters. In order to aid with diagnostics, the sequencer provides two rows of eight LED's: the bottom row of red LED's represents which thumper is being ordered to fire; the top row of green LED's notes which thumpers have returned a contact signal by firing.

Lastly, the sequencer provides a communications link between the monitoring computer and the receiver box. The monitoring computer is connected to the sequencer through an RS-232 standard nine-pin serial channel. This is converted to a TTL-level signal and relayed to a Hewlett-Packard fiber optic transmitter, which transmits the signal to the receiver box situated underwater through a fiber optic line. The receiver sends its data to the sequencer for relay to the monitoring computer the same way, in reverse. Fiber optic communications were chosen over electrical cables as the optimal system for lower noise in the communications line. The data rate used is 9600 baud, while the fiber optic line and the transmitter-receiver pair used are capable of handling forty kilobaud. The sequencer does not alter the signals between the receiver box and the monitoring computer except to convert RS-232 signals to fiber optic signals and vice versa.

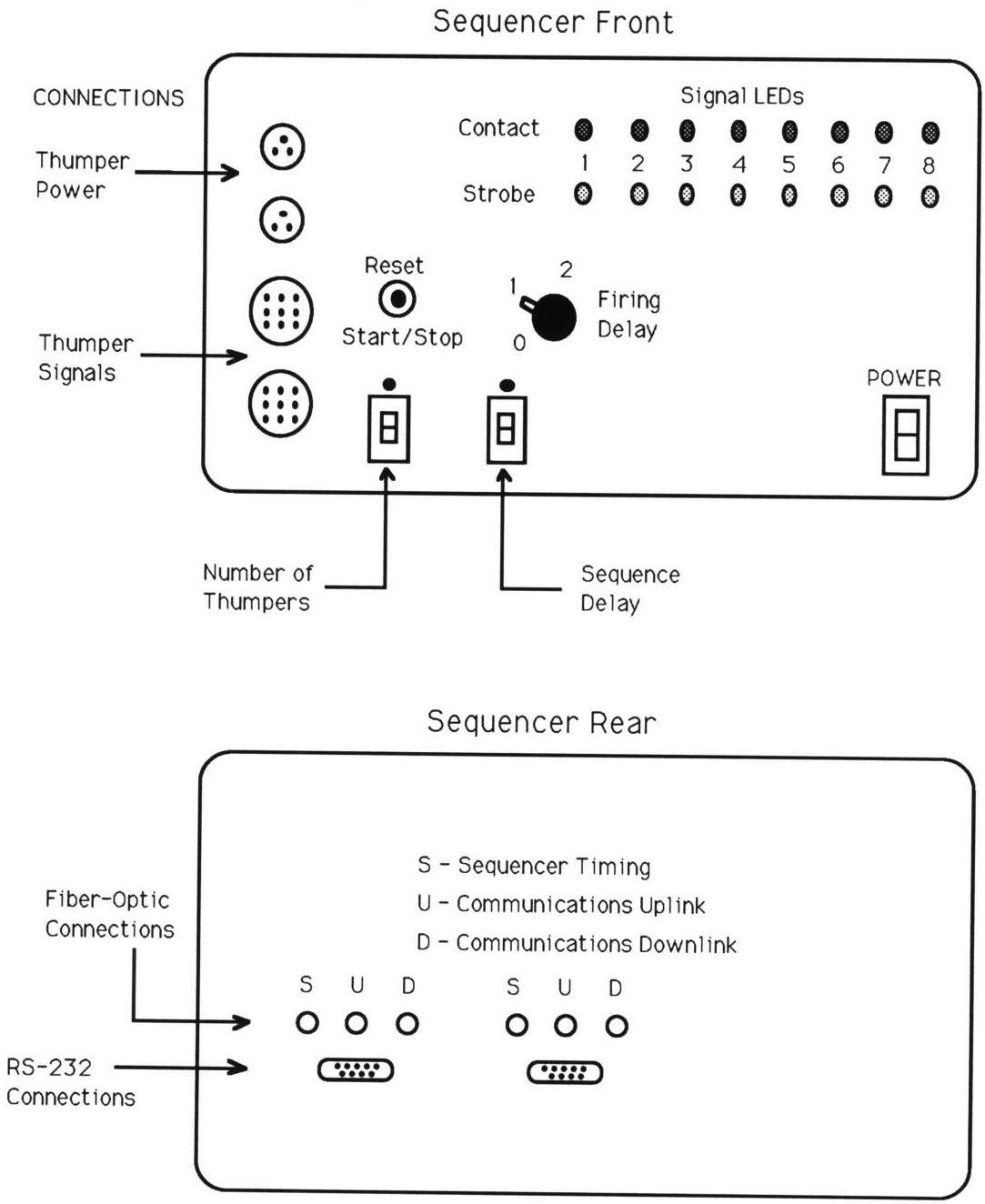


Figure 4. Sequencer Layout

2.2 The Receiver and Hydrophones

The sensors necessary to pick up the acoustic pulses generated by the thumpers are a set of four underwater microphones, known as hydrophones. The hydrophones chosen for the system were Brüel and Kjær Type 8103 Miniature Hydrophones (reference 3). Each consists of a piezoelectric ceramic microphone surrounded and sealed by chloroprene rubber. The hydrophone is capable of receiving signals of any frequency in the range 0.1 to 180 kHz, but has optimal reception at approximately 100 kHz. This compares favorably with the signal output from the thumpers. The hydrophone passes an incoming signal through a coaxial cable, which is connected to the inside of the receiver box through a watertight connection. The hydrophone itself rests in a small aluminum cage, to protect the sensor from accidental contact with the walls of the neutral buoyancy facility or other objects. A 69 cm aluminum rod is connected to the hydrophone cage, and provides stand-off distance for the hydrophone from the teleoperated robot to which it is attached. Thus, the body of a particular robot will be unable to completely block off a hydrophone from all the thumpers while the robot remains in the test area.

The receiver box shown in Fig. 5 contains all the necessary electronics to determine the straight-line ranges between any thumper and any hydrophone. Detailed circuit diagrams can be found in reference 2. An Ikelite™ box, sealed and pressurized to keep water from leaking in, contains the circuit cards used to perform the tasks needed to supply the monitoring computer with updated range information. A separate Ikelite™ box, also sealed and pressurized, holds gelled lead-acid batteries which power the receiver electronics while underwater. The battery box is connected to the main receiver box by a plastic tube holding the power wires. The maximum current drain of the receiver electronics in operation was measured to be 0.5 amps. A set of fully charged batteries can easily provide power for six to eight hours of underwater testing. A waterproof pushbutton on top of the receiver box allows the receiver to be

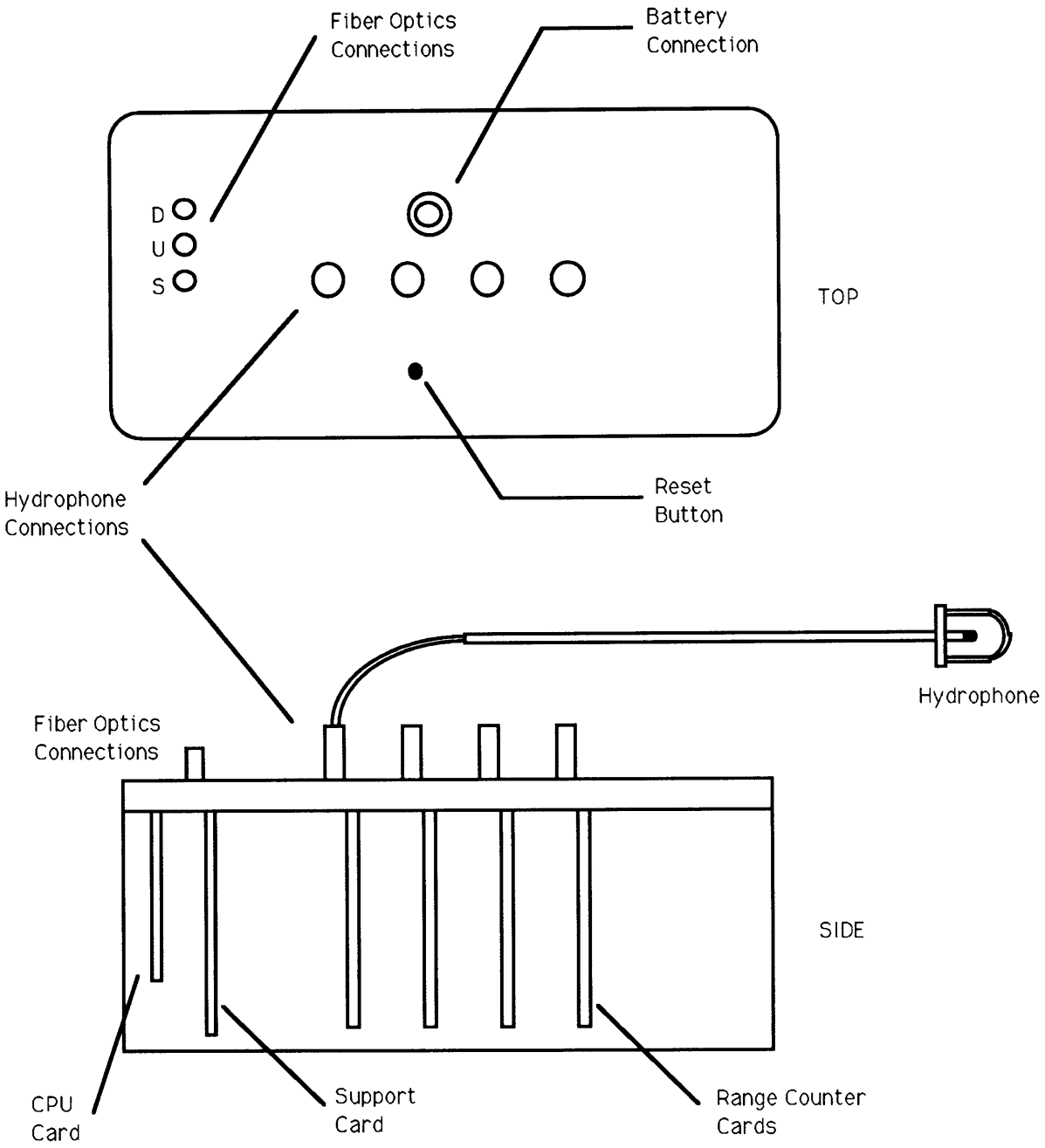


Figure 5. 3DAPS Receiver Diagram

reset underwater if something goes wrong and the problem can't be fixed from the surface.

The receiver's operation is governed by an Onset CPU-8088™ card, which provides good performance and functionality while requiring a maximum of 0.3 W to operate. More details regarding the CPU-8088 can be found in reference 4. The microprocessor is programmed in 8088 assembly language which is programmed onto an 8-kilobyte read-only memory (ROM) chip. The software source code is included in Appendix B. The other circuit cards contained in the box are a decoder/support card, and four range counter cards. The decoder/support card contains the electronic hardware to communicate through the fiber optic lines with the monitoring computer, circuits to decode the thumper identification and the contact signal, and the random-access memory (RAM) necessary to store the range values collected for a sequence of acoustic pulses as well as other important data needed to support the receiver software.

The final element of the receiver components is the set of four identical range counter cards. These cards, one connected to each hydrophone, provide the system with the raw data necessary to determine coordinates. In order to distinguish one card (and consequently, one hydrophone) from the remaining three, one of four switches on an eight-switch bank is flipped to 'on', while the rest are 'off'. Beyond this, each range counter card is identical, containing amplifiers and filters to pass a signal from the hydrophone to the TTL-level logic circuits, which subsequently determine the validity of the signal (see below) and halt the counter if the signal is valid or ignore the signal if deemed invalid. The counters have fifteen bits, which allows count values from 0 to 32767, permitting a maximum range of 25 meters using a 2.000 megahertz clock and a speed of sound of 1520 meters per second. The contact signal from the sequencer is passed directly to each of the range counter cards and is used to command the counters to start counting. The delay between the generation of the contact signal at the thumper and the start of the counters is on the order of nanoseconds and is

primarily due to chip-propagation times. An estimated upper limit of one microsecond delay would result in a constant offset error in the range values determined by the receiver of 1.5 millimeters. The accuracy of the system as determined in section 3 indicates that such a delay is insignificant. Once a valid signal from a hydrophone has arrived, the count value is placed on the data bus of the CPU-8088, along with flags detailing the validity of the answer shown.

The operation of the receiver box proceeds as follows: once powered on, or immediately following a reset, the microprocessor on the CPU-8088 card waits for a character, an ASCII carriage-return, to be sent from the monitoring computer on the surface. When it is received and if the receiver is responding correctly, an ASCII '+' is transmitted to the surface, indicating the receiver's status. The receiver then waits for a set of data to be sent. If the next character received from the monitoring computer is an ASCII 'A', the receiver prepares to accept the initialization data set from the surface. This set of data, as noted in the next section, consists of values which represent the number of thumpers in use, the x, y, and z coordinates of the operating thumpers, a value called the half-width (described below), and two values needed to convert the count value from the range counters to a range in millimeters and vice versa (described below). These numbers comprise a total of 53 initialization data bytes, assuming a full complement of eight thumpers is employed. The receiver then waits for the initialization data to be repeated, to verify the data previously received. If the second set of data received matches the first, an ASCII '+' is transmitted to the surface. Otherwise, an ASCII '-' is transmitted and the receiver waits for the initialization data to be sent and verified again.

Once a set of initialization data is received and verified the receiver again waits for a character to be transmitted by the monitoring computer. If the character received is an ASCII 'P', the receiver then waits for hydrophone coordinates to be transmitted. The receiver box expects to receive the x, y, and z coordinates of each of the four hydrophones, plus a byte containing a checksum of the previous 24 bytes, for a total

of 25 bytes of data. If the received checksum matches that calculated by the receiver, an ASCII '+' is transmitted to the monitoring computer. Otherwise, an ASCII '-' is transmitted, and the receiver box waits for the hydrophone data and checksum to be sent again.

Once a verified set of hydrophone coordinates has been received, the receiver box calculates the estimated ranges from each thumper to each hydrophone for the next thumper sequence, based on the set of thumper coordinates received in the initialization data set and the latest set of hydrophone coordinates received. The estimated ranges are converted from millimeters to count values using the ratio of the speed of sound (in millimeters per second) to the rate of the clock used to drive the counters (2.000 megahertz). Because the 8088 operates only with integers, the ratio is broken up into two numbers, a numerator and denominator. The values used for the experiments were 19 and 25 for the numerator and denominator, respectively. This corresponds to a speed of sound of 1520 meters per second. The estimated range is multiplied by the denominator and then divided by the numerator to be converted to the count value estimated for a particular hydrophone/thumper combination. The value called the half-width is then added to and subtracted from the estimated count value, limited by the maximum or minimum possible count value, to arrive at the maximum and minimum estimated count values for each hydrophone/thumper combination. This provides a window of count values into which a valid count is expected to fall. Once all the possible hydrophone/thumper window limits are calculated and stored, the range matrix for storing the acquired ranges is cleared and then a set of hydrophone/thumper ranges is awaited.

The reasoning behind determining the expected count windows is to provide a masking of signals that occur significantly before or significantly after the estimated arrival time as determined by the estimated range. The acoustic pulses generated by the thumpers will reflect off the walls of the neutral buoyancy facility. Because of this it is possible that a reflection of one acoustic pulse could be admitted as a valid

signal, either by arriving slightly later than the original, correct, pulse and competing with it for acceptance, or by arriving ahead of the pulse generated by a subsequent thumper. With both maximum and minimum expected count values defining a window of acceptable count values, these problems are reduced. The first signal arriving within the window is assumed to be correct. Signals arriving early are completely ignored, while signals that arrive late are used as a flag for the microprocessor to increase the window size. The range returned for the signal will be zero in either case, but the window of valid count values will be increased by a factor of two for that particular hydrophone/thumper combination if the signal arrived late. If the receiver subsequently acquires a valid signal for that combination it will cut the window size in half, down to the size allowed by the user's choice of half-width. The half-width value used for the experiments in this thesis was set at 5000, representing a range value of 3800 millimeters. This was determined empirically through testing the system at various half-widths ranging from 50 to 10000 (38 to 7600 millimeters).

When the sequencer orders a thumper to fire and the thumper firing is detected, the sequencer transmits a contact signal to the receiver following the thumper's identification number. The contact signal is shunted straight from the fiber optic line to the range counters, which start running. The receiver's microprocessor never intercepts the contact signal directly, so as to not increase the time delay between a thumper firing and the range counters' starting to count. The receiver software repeatedly tests the range counter cards to determine if they have started counting. If they have, the thumper identification is loaded onto the 8088's data bus from the support card where it had been stored when sent down from the sequencer. The thumper identification is used to set the count windows for each range counter. The maximum and minimum values are loaded into each range counter card, and then the receiver waits for the range counters to cease counting. When all four have finished, the microprocessor reads in the count values and checks the flags set by the range counter cards for the validity of the count value. If the pulse arrived at the hydrophone late

(after the counter had passed the maximum expected count value) or not at all, the ranges returned to the monitoring computer will be zero. The range counters stop counting upon overflow. Similarly, if the only pulse arrived before the counter had passed the minimum expected count value, the range returned will be zero. If a pulse was received within the window of expected count values, the value determined is converted to millimeters using the speed-of-sound numerator and denominator provided in the initialization data and the range is stored in RAM. When all four ranges have been acquired for a particular thumper, the receiver compares the number of the thumper that just fired with the number of the last thumper in the sequence as given by the number of thumpers sent down in the initialization data. If they match, the receiver sends all the range data, along with a checksum, to the monitoring computer on the surface. For eight thumpers and four hydrophones this is a total of 64 bytes of data, plus a checksum. The receiver waits for the monitoring computer to verify the data, expecting an ASCII '+' if received correctly or a '-' if not, which requires that the data be sent again. When the receiver is sent an ASCII '+', it then waits for another character to be sent down from the monitoring computer, either an ASCII 'P' indicating another sequence of ranges to be collected, or an 'A', indicating a new initialization data set.

The problem of requiring the last thumper in the sequence to fire stems from the fact that the receiver needs to know when a full set of data has been acquired. The only way to insure this is by recognizing the last thumper of a sequence and transmitting the range data to the surface after its pulse has been received. Thus, if the final thumper of a sequence never fires, the receiver will collect data for every contact signal it receives, writing over the previous range values with every new one that it determines, but it will never send the data to the surface.

2.3 The Monitoring Computer

The basis of 3DAPS stems from a simple equation:

$$(x_h - x_t)^2 + (y_h - y_t)^2 + (z_h - z_t)^2 = r_{ht}^2 \qquad (2.3-1)$$

where $(x, y, z)_h$ denote hydrophone coordinates and $(x, y, z)_t$ denote thumper coordinates.

The system relies on range information (r_{ht}) between hydrophone h and thumper t as determined through the actions of the sequencer and the receiver. By using the locations of three independent thumpers, and the three independent ranges to them, there will be three independent equations in three unknowns. This is a solvable system of equations. Since the range measurements have associated errors, however, more thumpers are employed than the minimum of three necessary to achieve a solution to (2.3-1). It then becomes necessary to solve an over-determined system of simultaneous equations. This is the main function of the monitoring computer, either an IBM PC/AT or an Apple Macintosh II. This section describes the software used to run the 3DAPS system and to determine the positions of the four hydrophones. A listing of the code used in the monitoring computer (IBM) appears in Appendix B.

The tasks of both the receiver and the monitoring computer are closely linked. The monitoring computer's job is to initialize the receiver, acquire the uplinked ranges, calculate the coordinates based on those ranges, and send them back to the receiver. It also displays the data to the user. A description of the actions of the software used follows.

The program used in these experiments required the user to input the initialization data needed to begin operations. As mentioned in the previous section, this includes a value called the half-width, the speed-of-sound numerator and denominator, the number of thumpers and their locations, and the initial locations of the hydrophones. The software requests the user to verify the initialization data entered and if valid,

primes the receiver by transmitting an ASCII carriage-return character. If this is acknowledged, the monitoring computer then transmits an ASCII 'A', followed by the initialization data, sending the data a second time immediately afterward. If an ASCII '+' is then received in acknowledgement, the program continues. If an ASCII '-' is received the initialization data is sent twice more.

The initialization data is sent twice as a greater safeguard against data corruption than the addition of a checksum byte. Because the initialization data is sent before the experiment is begun, enough time is available to transmit the data twice.

Next, the monitoring computer transmits an ASCII 'P', followed by the current set of hydrophone coordinates and a checksum. This initiates the loop the computer will execute while the experiment is in progress. The program again waits for an acknowledgement, and again an ASCII '+' signals "ok", while a '-' causes a repeated transmission of the data set. If successfully acknowledged, the computer will wait for its serial input buffer to accumulate range data transmitted from the receiver. Assuming the use of eight thumpers, this will amount to 65 data bytes, including a checksum. The program computes its own checksum, compares it to that transmitted by the receiver, and acknowledges the receiver with an ASCII '+' or '-', depending on the result of the comparison.

The computer then calculates the coordinates of the hydrophones based on the range data transmitted by the receiver. The algorithm employed follows a least-squares fit utilizing the Newton-Raphson method of function minimization to reduce the error between the hydrophone/thumper ranges as based on an estimated hydrophone position, and the ranges as measured by the 3DAPS receiver. The most recently calculated hydrophone coordinates are used as the initial guess value for the algorithm. The coordinate calculations code embodies these equations:

$$\Delta x_{htk} = x_{hk-1} - x_t$$

$$\Delta y_{htk} = y_{hk-1} - y_t \qquad (2.3-2)$$

$$\Delta z_{htk} = z_{hk-1} - z_t$$

$$d_{htk} = \sqrt{(\Delta x_{htk})^2 + (\Delta y_{htk})^2 + (\Delta z_{htk})^2} \qquad (2.3-3)$$

d_{htk} represents the estimated range between thumper t and the estimated position of hydrophone h for the kth iteration.

$$e_{htk} = d_{htk} - r_{ht} \qquad (2.3-4)$$

$$c_{hk} = \sum_t (e_{htk})^2 \qquad (2.3-5)$$

c_{hk} is the "cost": the sum of the squared errors over all the thumpers.

$$\begin{aligned}
\frac{\partial}{\partial x}(c_{hk}) &= \sum_t \left(\frac{\Delta x_{htk} e_{htk}}{d_{htk}} \right) \\
\frac{\partial}{\partial y}(c_{hk}) &= \sum_t \left(\frac{\Delta y_{htk} e_{htk}}{d_{htk}} \right) \\
\frac{\partial}{\partial z}(c_{hk}) &= \sum_t \left(\frac{\Delta z_{htk} e_{htk}}{d_{htk}} \right)
\end{aligned} \qquad (2.3-6)$$

$$\begin{aligned}
\frac{\partial^2}{\partial^2 x}(c_{hk}) &= \sum_t \left(\frac{(\Delta x_{htk})^2 r_{ht}}{(d_{htk})^3} + 1 - \frac{r_{ht}}{d_{htk}} \right) \\
\frac{\partial^2}{\partial^2 y}(c_{hk}) &= \sum_t \left(\frac{(\Delta y_{htk})^2 r_{ht}}{(d_{htk})^3} + 1 - \frac{r_{ht}}{d_{htk}} \right) \\
\frac{\partial^2}{\partial^2 z}(c_{hk}) &= \sum_t \left(\frac{(\Delta z_{htk})^2 r_{ht}}{(d_{htk})^3} + 1 - \frac{r_{ht}}{d_{htk}} \right)
\end{aligned} \qquad (2.3-7)$$

The first and second derivatives of the cost with respect to each axis are computed for use in the Newton-Raphson calculations. The next equation calculates the gradient of the cost.

$$\nabla = \left(\frac{\partial}{\partial x}(c_{hk}) \right)^2 + \left(\frac{\partial}{\partial y}(c_{hk}) \right)^2 + \left(\frac{\partial}{\partial z}(c_{hk}) \right)^2 \qquad (2.3-8)$$

$$\text{if } \nabla > c_{\text{tol}} \text{ then}$$

$$x_{hk} = x_{hk-1} - \frac{1}{2}\left(\frac{\frac{\partial}{\partial x}(c_{hk})}{\frac{\partial^2}{\partial^2 x}(c_{hk})}\right)$$

$$y_{hk} = y_{hk-1} - \frac{1}{2}\left(\frac{\frac{\partial}{\partial y}(c_{hk})}{\frac{\partial^2}{\partial^2 y}(c_{hk})}\right) \qquad (2.3-9)$$

$$z_{hk} = z_{hk-1} - \frac{1}{2}\left(\frac{\frac{\partial}{\partial z}(c_{hk})}{\frac{\partial^2}{\partial^2 z}(c_{hk})}\right)$$

The estimated position of the hydrophones are adjusted according to Newton-Raphson's method based on the gradient value.

The symbols used in the above equations are listed below:

$h =$ hydrophone index
$t =$ thumper index
$k =$ iteration index
$(x, y, z)_{hk} =$ coordinates of hydrophone h during iteration k
$(x, y, z)_t =$ coordinates of thumper t
$c_{\text{tol}} = 8.0$, gradient tolerance
$r_{ht} =$ range as determined by 3DAPS between hydrophone h and thumper t

The value for the gradient tolerance used was taken from the receiver software in ref. 1. The system described in the reference performed all the necessary calculations within the receiver box, displaying the results on an LCD screen underwater. However, using four hydrophones instead of one, the time delay due to the coordinate calculations increased, and so it was decided that a faster method of performing the calculations would be to send the data to a computer at the surface. The receiver software in ref. 1 allowed a maximum of ten iterations to be run to calculate each hydrophone's coordinates, as does the software used for these experiments.

The monitoring computer software utilizes equations 2.3-2 through 2.3-9 to determine for each hydrophone its x, y, and z coordinates with the full set of range data for each hydrophone sent up from the receiver. Zero ranges are ignored. Then, anticipating that an invalid range value might have been accepted by the receiver, the software recalculates the coordinates for each hydrophone eight times, each time ignoring a

different hydrophone/thumper range. The nine sets of coordinates determined for a hydrophone are compared, and the set with the lowest "cost" as defined by c_{hk} in equation 2.3-5 is selected as the coordinates of the hydrophone for that particular set of range values. The coordinates are then transmitted to the receiver, and assuming a valid acknowledgement is returned, the computer awaits the next set of ranges.

The calculations for determining vehicle attitudes were performed using the algorithm found in ref. 1.

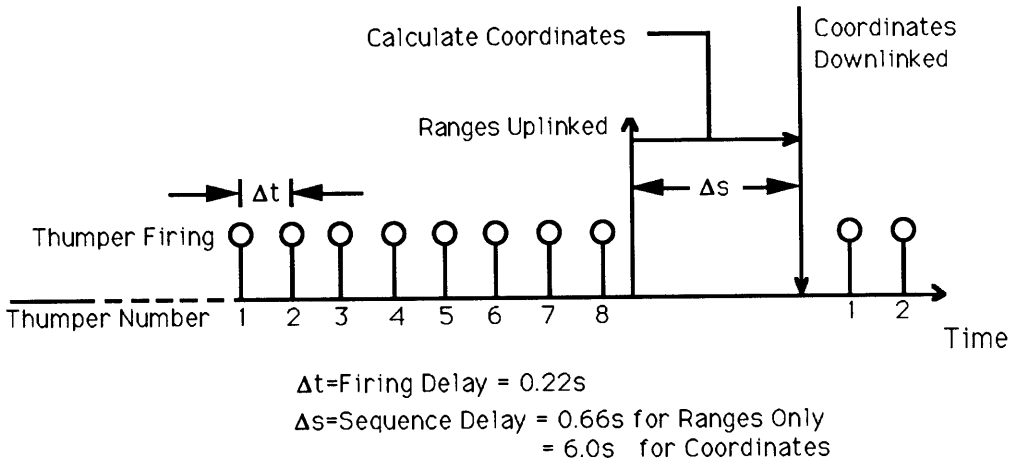
The software required to run on the monitoring computer was designed and developed on an IBM PC/AT running DOS version 3.2, using Microsoft™ C version 5.1 . Extensive use was made of the Greenleaf™ libraries version 3-2.21 for the serial communications routines necessary to communicate with the receiver. While testing the system's limitations, it became apparent that an IBM PC/AT running at a six megahertz clock speed would not calculate coordinates using the above algorithm fast enough for useful results. The chronology of the system operation is shown in figure 6.

On average, the IBM PC/AT required between seven and eight seconds to determine the coordinates of the hydrophones based on a full set of range data. Thus, it was decided to use the IBM to display and record the range values and the time at which the data was uplinked in real time. A second program was developed to read in the range data previously stored on the IBM and determine the coordinates after the experiments had been concluded.

In contrast, an Apple Macintosh II was also configured to act as the monitoring computer, with results indicating that the same software ported to the Macintosh would run significantly faster. The chronology of events with the Mac II as the monitoring computer is shown in figure 7.

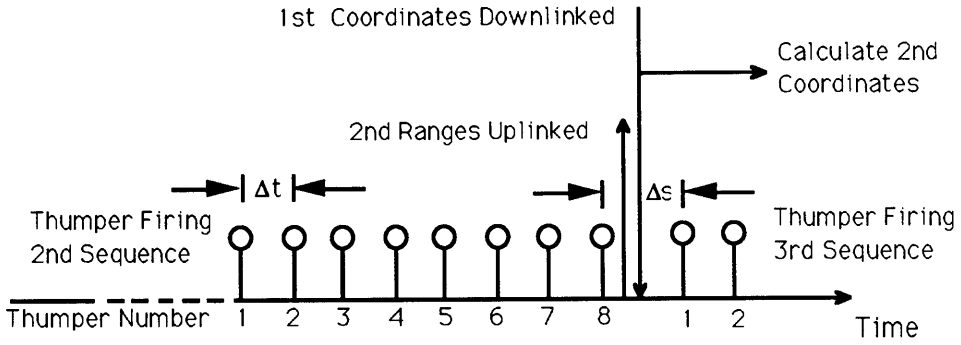
The Mac II calculated the coordinates from a full set of ranges fast enough using the algorithm to permit some speeding-up of the sequencer. By calculating the coordinates from one set of ranges while the sequencer was transmitting the next, the

sequence delay was reduced and the system as a whole could run faster. The calculated coordinates would lag the actual position by one sequence. Tests showed that by using the Mac II in this fashion, the total time per sequence was reduced to 1.6 seconds, on the average, allowing the full coordinates and attitudes calculation code to be utilized in real time. However, the experiments in this thesis were conducted using an IBM PC/AT programmed as described above, recording the ranges in real time and performing calculations after the fact. The coordinates downlinked to the receiver were those as input by the user initially.



Δt=Firing Delay = 0.22s
Δs=Sequence Delay = 0.66s for Ranges Only
= 6.0s for Coordinates

Figure 6. System Event Chronology: IBM AT



Δt=Firing Delay = 0.22s
Δs=Sequence Delay = 0.44s for Coordinates

Figure 7. System Event Chronology: Apple Macintosh II

3.0 The 3DAPS Experiments

The capabilities and performance of the 3DAPS system are best demonstrated in the environment for which the system was designed. Experiments were conducted in a neutral buoyancy facility to establish the capabilities of the system, and to show the possible endeavors to which the system may be usefully applied.

To fully demonstrate the capabilities and limitations of the system, two tests were designed, developed, and implemented for the system. The first set was a series of static tests, designed to show that in an ideal setting, the system can operate satisfactorily. The behavior of the system in this category of tests, performed at both the MIT pool and the Marshall Space Flight Center, is detailed in Section 3.1.

The second category of tests was the controlled dynamic experiment, discussed in Section 3.2. This section illustrates the capability of the system in a less-than-ideal operation, that of determining hydrophone positions and the consequent vehicle attitude with the hydrophones' positions not fixed during measurements. This experiment has a great bearing on the use of the system in conjunction with a feedback control system governing the operation of a robot.

The final sections of this chapter discuss applications of the present system to other projects in the SSL. These include trajectory tracking for the Beam-Assembly Teleoperator (BAT) and the Multimode Proximity Operations Device (MPOD), as well as determining the reach envelopes of humans both in and out of pressure suits.

For the static experiments, the reach envelope tests, and the vehicle trajectory experiments, the IBM software simply recorded ranges and times. For the controlled dynamic experiments, a circuit card was built to interface with an optical encoder, and the software was consequently augmented to read and store encoder values along with the time and range data. A diagram of the encoder interface circuit can be found in Appendix E. The non-realtime calculation software was then modified to read in either format of data and produce coordinates and attitudes of the objects to which

the hydrophones were attached. Listings of all the codes used for these experiments are included in Appendix B.

3.1 Static Tests

In order to determine the validity of the system's data a set of tests was devised to demonstrate the its operation and capabilities in a static configuration. The hydrophones were held immobile with respect to the thumpers so that the ranges and consequently the calculated coordinates would be constant. With such a predetermined setup, the system can be evaluated and compared with independent position and range. This forms the basis for determining the validity of the system design.

A total of five static tests were conducted. Four were run at the MIT pool, and the fifth was conducted at the Marshall tank. In the tests the hydrophone rods were bolted to a piece of PVC, in a tetrahedral arrangement as shown in figure 8. The positions of the hydrophones relative to a coordinate system assigned to the base of the PVC are listed in Appendix C. The PVC was connected to the Instrumented Beam System (IBS) stand (reference 5). The stand consists of a circular aluminum base bolted to a cylindrical pedestal, to which the PVC base was bolted. This gave the hydrophones a static platform from which the tests could be successfully conducted. The setup was placed at different locations in the MIT pool within the test area, and was placed at one location for a test at the Marshall tank to prove system validity in that environment. The test area at each facility was the volume of water defined by the intersection of the thumper signal paths. Figure 9 shows the setup and the placement of a static test at the MIT pool.

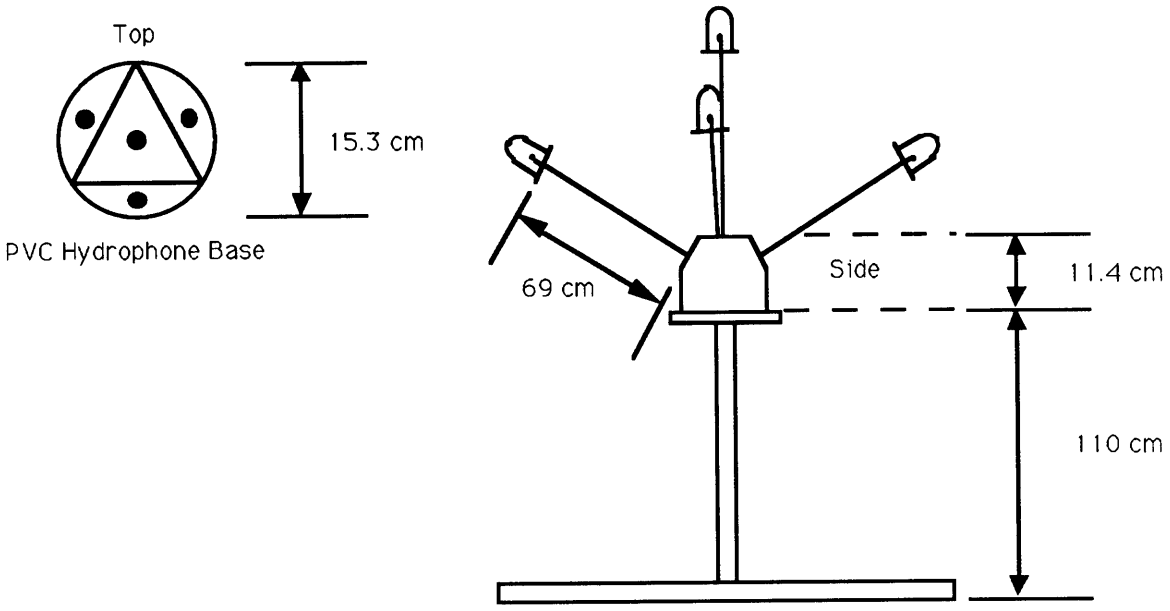
In each experiment, the platform was set up and placed arbitrarily within the test area. Using a tape measure, divers then measured the straight-line ranges of each of the hydrophones to each of the thumpers, and the coordinates of each of the hydrophones in the assigned coordinate frame (coordinates could not be directly

measured at the Marshall tank). Once the measurements had been completed, the test area was cleared of extraneous objects and the system was allowed to collect data for periods of time ranging from five to fifteen minutes. Range data was collected during the entire period and recorded along with the time at which each range data set was acquired by the monitoring computer. The ranges were then processed afterwards to determine the average ranges for the sample and their standard deviations, plus the coordinates and their standard deviations as calculated by the algorithm described in Section 2.3.

Figure 9 shows the relationship between the ranges as measured by the 3DAPS system at the MIT pool and the ranges as measured directly by the divers. A linear curve fit was performed on the sample, showing a correlation coefficient of 0.999 and a nearly perfect unit ratio between the measured and acoustic ranges. The offset constant is likely due to errors made by the divers in measuring the ranges. Figure 10 shows the relationship of the coordinates as calculated by the system at the MIT pool and the coordinates as determined by direct measurements. A linear curve was fit to the data and the resulting correlation coefficient was 0.998. Again, a nearly perfect ratio was noted between the measured and acoustic values for the coordinates. Figure 11 shows the results for the range measurements at the Marshall tank, with results similar to those at the MIT pool.

In each static case, the range data acquired by both acoustic and direct measurement proved the validity of the system. The data used for the graphs can be found in tabular form in Appendix D. In each case, the standard deviations of the range samples were less than 1% of the range determined, at the most about 9 cm for a 1450 cm range. The average standard deviation of the five static cases was 2.9 cm.

Also, the calculated coordinates behaved as well as the ranges for each case. With the exception of the z-axis coordinates, the standard deviations of the coordinates were similar to those of the ranges, within 1% of the coordinate being analyzed. The standard deviations of the z-axis coordinates in each case were significantly larger



Hydrophone Cluster and IBS Support Structure

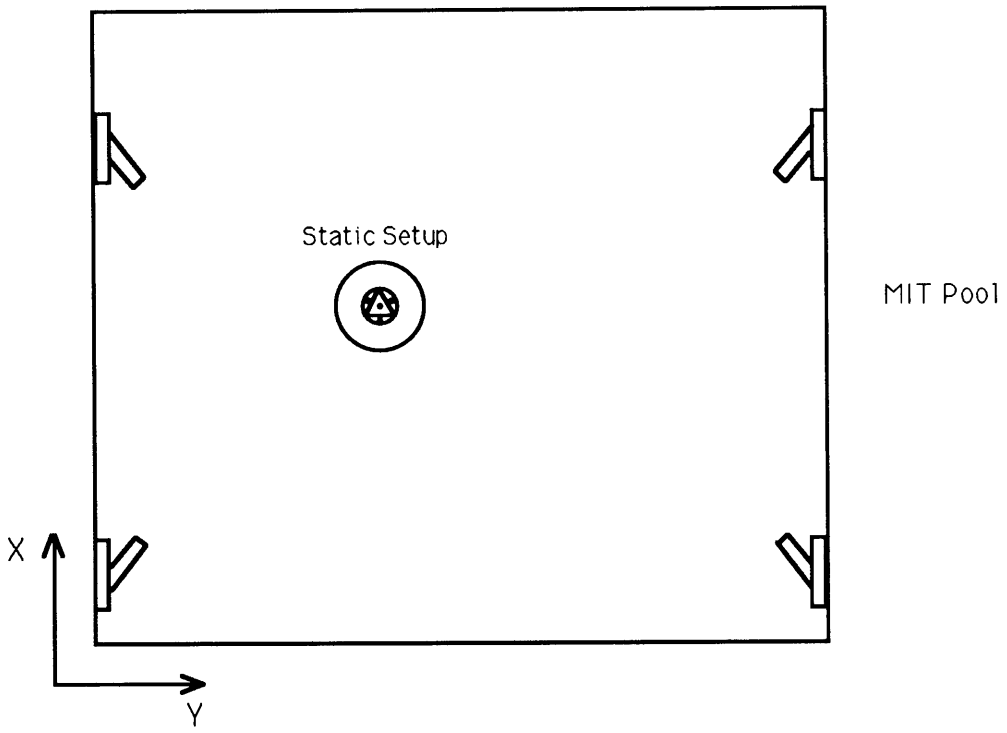


Figure 8. Static Test Setup

than those of the x and y coordinates, with an average of 8.8 cm for z as compared to an average of 3.8 cm for x and an average of 2.8 cm for y. This can be attributed to the significantly larger difference in the thumper x and y coordinates as compared to the difference in the thumper z coordinates. This difference gives the x and y coordinates a larger baseline over which to be calculated, as the thumpers are more "independent" in the x- and y-axes by being farther away from each other than in the z-axis.

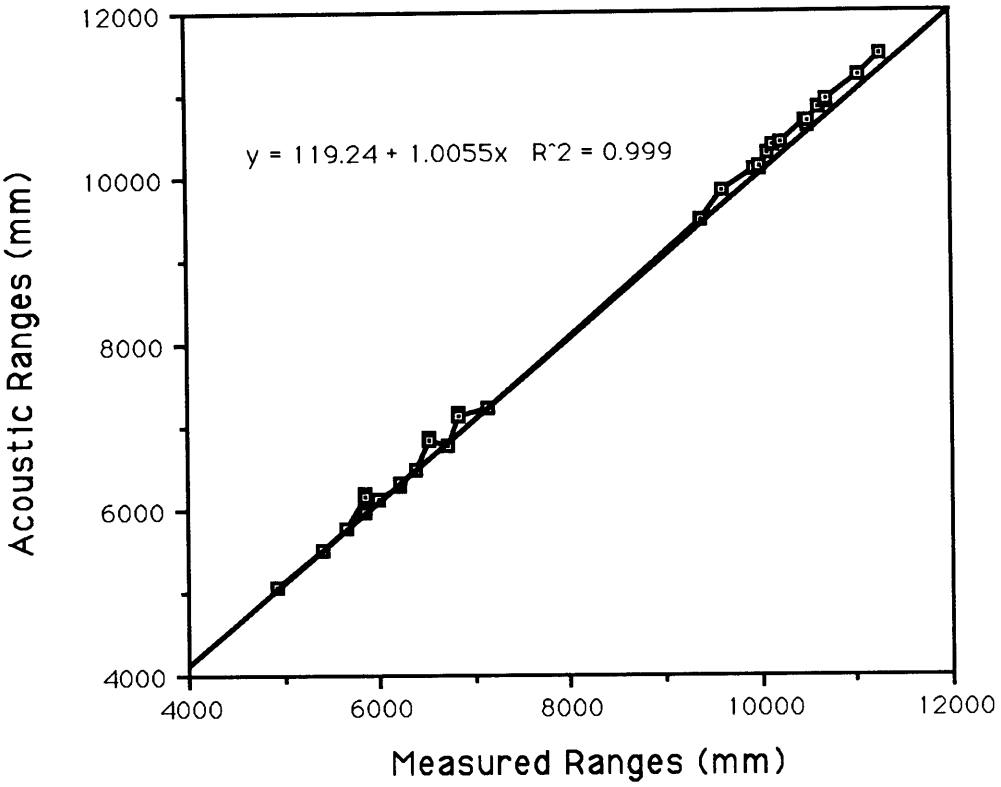


Figure 9. MIT Pool Range Data

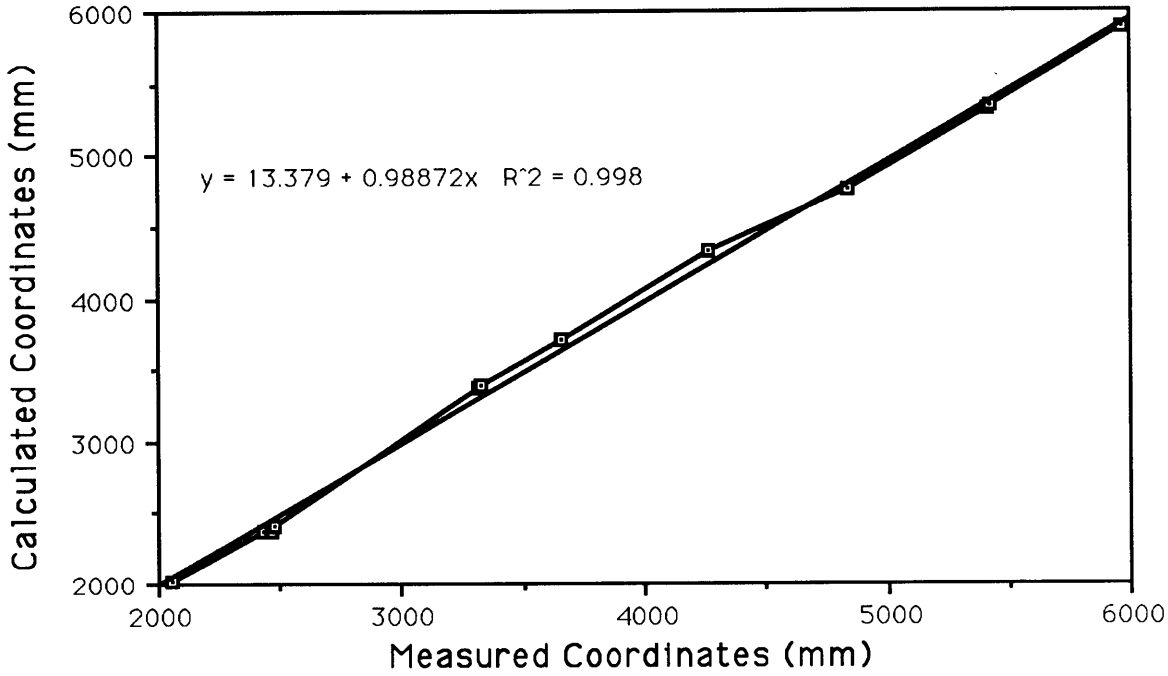


Figure 10. MIT Pool Static Test Coordinates

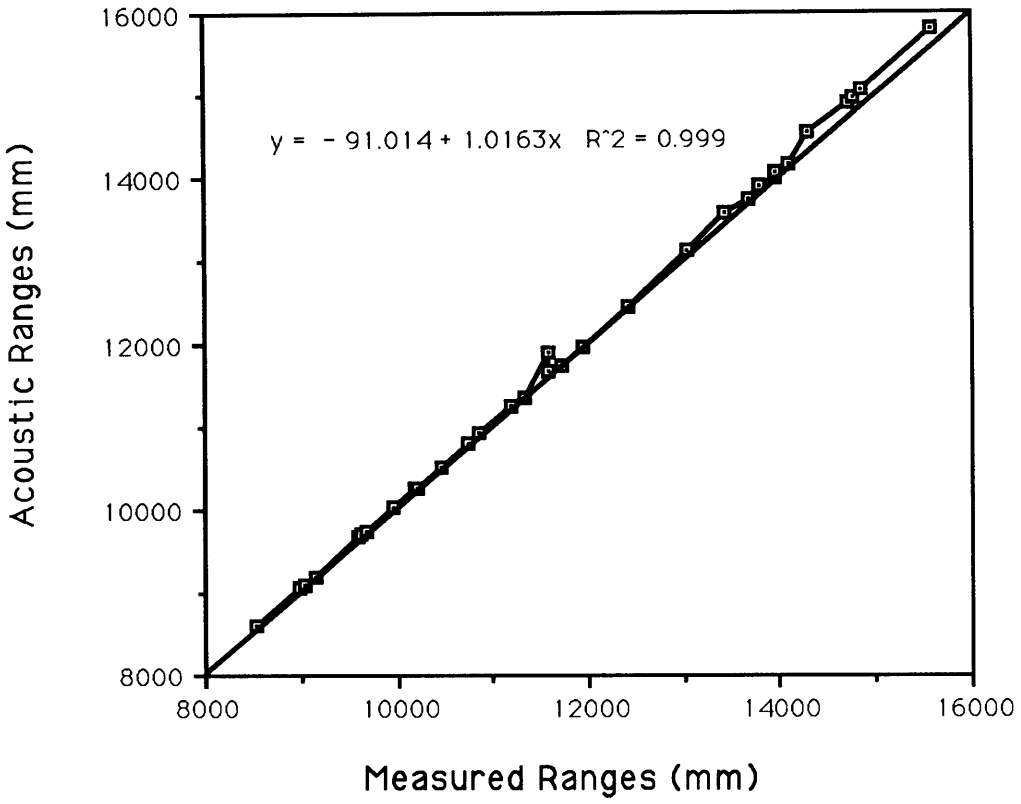


Figure 11. Marshall Tank Range Data

3.2 Controlled Dynamic Tests

Through the static tests, it was shown that the system concept and design does work. However, in order to be assured of the usefulness of the system in non-static operating conditions, dynamic tests in a controlled environment were devised and executed in the MIT pool. Figure 12 shows the setup of the equipment and the placement of the apparatus within the test area.

An aluminum box-beam was placed at the bottom of the pool. A low-friction cart made of Delrin and teflon-covered aluminum was constructed to run on the beam and to connect to the PVC base as used in the static tests. A cable was attached to the cart and ran along the beam, through a pulley at one end, and up to a Sumtak optical encoder (HEDS-2000) sitting at the surface. The encoder was connected to a circuit card (see Appendix E) placed in the IBM PC/AT which kept track of the encoder reading through the use of a Hewlett-Packard General Purpose Motion

Control Integrated Circuit (HCTL-1000) which used a 24-bit counter to record the encoder position. The beam was held on the bottom with lead weights, and the end of the cable at the surface was also held taut by lead weights.

Complete static tests were performed at each end of the beam. Divers directly measured the ranges between each thumper and hydrophone, as well as the coordinates of each hydrophone. The system was allowed to run, collecting range, time, and encoder information at both ends of the beam. Also, partial static tests — range and encoder measurements by the system only — were performed at different points along the beam. A diver would move the cart by roughly one-fourth the traverse and hold the cart in place while static range and encoder data was gathered at that point. Thus, static range data was stored for five points along the beam: the two endpoints, and three points spread evenly in between. This operation provided static tests along the beam, locating sets of coordinates at various encoder positions.

Lastly, the equipment was set up to allow the hydrophones to move while the system was in operation. As a diver held the cart at the far end of the beam, varying amounts of weight were added to the surface end of the cable, and after allowing the system to acquire several sets of static range data, the cart was released. The PVC base supporting the hydrophones would then be in motion while the system was attempting to measure the ranges between the hydrophones and the thumpers, and the encoder values would be read every time a set of range values was uplinked from the receiver. The experiment was run a number of times, using weights from four to sixteen pounds to accelerate the cart along the track.

The analysis of the data collected by the monitoring computer showed that while the hydrophones are in motion, incorrect hydrophone coordinates will be calculated from the ranges determined by the receiver. This is due to the fact that the hydrophones move between thumper firings, causing ranges from all the thumpers employed to be measured at different hydrophone positions during the course of a sequence of pulses. At best, this will result in a set of hydrophone coordinates being

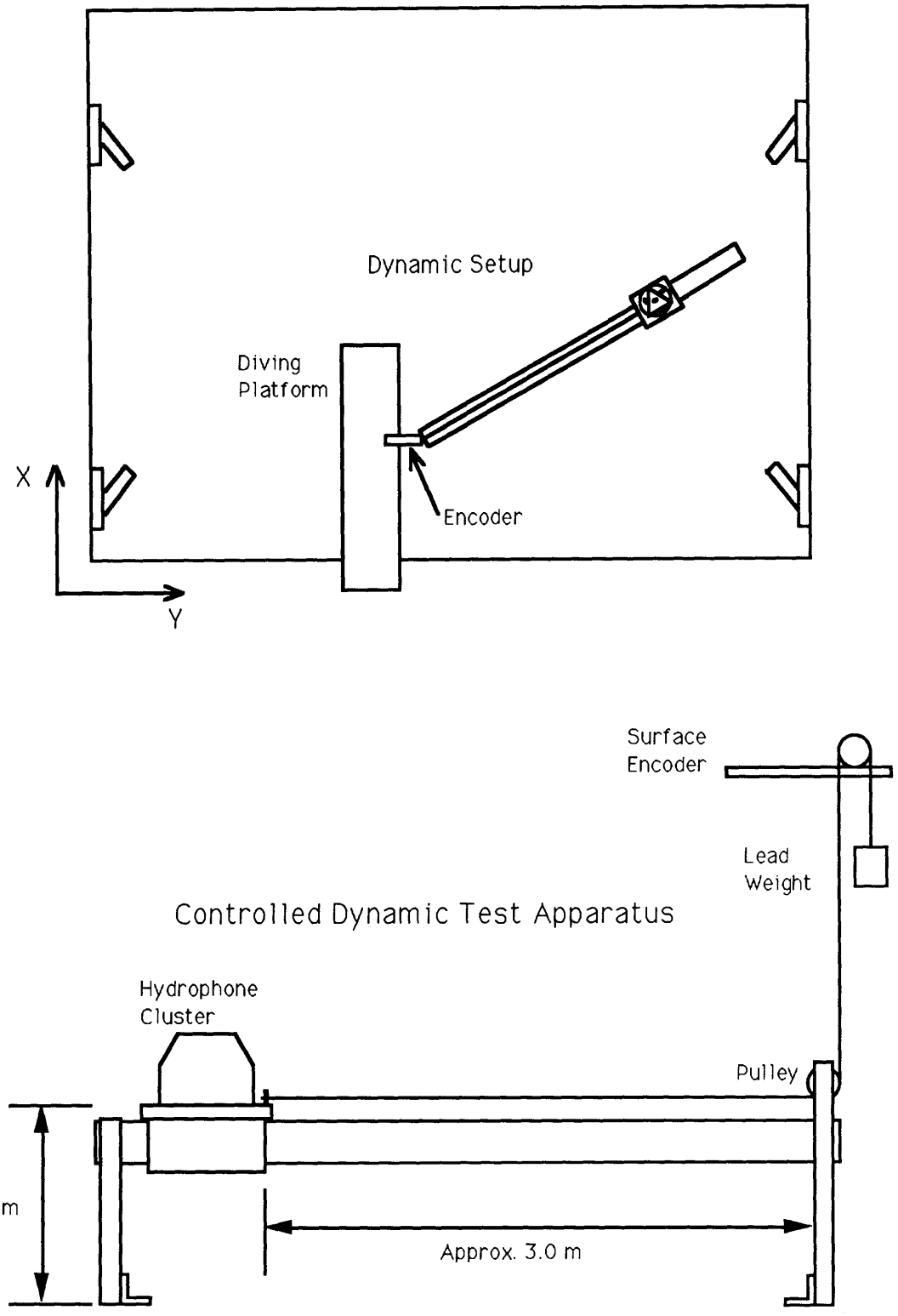


Figure 12. Dynamic Experiment Setup

calculated as somewhere between the point the hydrophones occupied when the first thumper pulse was received and the point occupied when the last thumper pulse was received.

The static tests at each end of the beam provided a relationship between the encoder values read along the track and the coordinates of the hydrophones when the cart was at a particular encoder position. This relationship was determined as follows. The total distance traversed was calculated by determining the distance travelled by the hydrophones based on their beginning and ending coordinate values. This was then compared to the change in encoder values from one end of the beam to the other, as measured in encoder counts and converted to millimeters. The two distance measurements agreed within experimental tolerances, and therefore a relationship governing the coordinates of the hydrophones as determined by an encoder value was deemed valid. Equation 3.2-1 shows the relationship between an arbitrary encoder value and the associated hydrophone coordinates.

$$
\begin{aligned}
x_e &= \frac{e - e_{\text{far}}}{e_{\text{near}} - e_{\text{far}}}(x_{\text{near}} - x_{\text{far}}) + x_{\text{far}} \\
y_e &= \frac{e - e_{\text{far}}}{e_{\text{near}} - e_{\text{far}}}(y_{\text{near}} - y_{\text{far}}) + y_{\text{far}} \\
z_e &= \frac{e - e_{\text{far}}}{e_{\text{near}} - e_{\text{far}}}(z_{\text{near}} - z_{\text{far}}) + z_{\text{far}}
\end{aligned}
\qquad (3.2-1)
$$

where

e = Arbitrary Encoder Value
e_{near} = Encoder Value, Near End of Beam
e_{far} = Encoder Value, Far End of Beam
$(x, y, z)_{\text{near}}$ = Coordinate, Near End of Beam
$(x, y, z)_{\text{far}}$ = Coordinate, Far End of Beam
$(x, y, z)_e$ = Coordinate, Determined from Encoder Value

Given the above relationship, the coordinates of the hydrophones were determined relative to the encoder values during the dynamic portion of the experiment.

Figures 13 shows the ratio of the calculated coordinates to the encoder-determined coordinates, plotted against the encoder position in count values along the track during both static and dynamic portions of the experiment. The ratio between encoder count values and meters length was determined as 52200 counts per meter. Figure 13 shows the x, y, and z coordinate ratios for hydrophone zero during the static tests. The cart and the PVC base were held in position at five points, including the two endpoints, to produce the coordinates for this graph. As is evident, the ratios for the three coordinates are nearly unity, indicating that the coordinates as calculated by the system are nearly identical to the coordinates as determined from the encoder values along the beam, given the hydrophone coordinates as determined in the static tests at either end. The calculated z coordinates are noticeably different from those as determined from the encoder position, like the z coordinates as determined in the static tests.

Figures 14 through 19 show the plots of calculated x, y, and z coordinates versus the coordinates as determined by the encoder position for hydrophones zero, one, and two during two dynamic tests. Hydrophone three is not used because of a malfunction within the receiver giving erroneous data. Ideally, the calculated coordinates should match the encoder-determined coordinates. The velocity of the cart in test one was determined to be 0.57 meters per second; the velocity in test two was 0.51 meters per second. As measured by the encoder, the total traverse for each test was 3.44 meters. This compares favorably with the distance as determined from the hydrophone coordinates at both ends, which was calculated as 3.4 meters, a 1.2% difference. In both tests, the coordinates as calculated by the system are significantly different from those as determined by the encoder values. In figures 14 through 19 it can be seen that while the hydrophones are in motion the system cannot exactly calculate coordinates. In both dynamic tests the calculated coordinates are unreasonable while the hydrophones are in motion. In the worst cases there is a difference of 50% between the calculated and encoder-determined values of the x coordinate. The graphs show

that the calculated values at the endpoints are comparable to those determined by the encoder position because the cart at both points was at rest.

One possible source for the coordinate discrepancies apparent in the two dynamic tests is the proximity of the hydrophones to both the floor of the neutral buoyancy facility and one wall. Reflections of thumper signals, as well as hydrophones being out of the thumper signal area, could have produced ranges that were inaccurate, which would lead to incorrect coordinates.

In order to more fully define the velocity dependence of the coordinate error, a simulation was developed to completely model the controlled dynamic experiment. The simulation code can be found in Appendix B. Given the encoder values and static coordinates at each end of the beam, this simulation models the dynamic experiment by assuming a velocity and calculating the hydrophone coordinates for each acoustic pulse by determining the encoder value at the time each pulse is received. The ranges between the four hydrophones and each pulse-generating thumper were determined for each set of pulses while the hydrophones were in motion along the beam. The coordinates were then calculated as usual, and a profile of the theoretical coordinate errors versus velocity was generated. The resulting profile is shown in Fig. 20. This figure represents the average error per thumper sequence simulated for velocites ranging from 0.1 meters per second to 2.0 meters per second. The simulation used the static data acquired for the dynamic tests to relate encoder position along the beam. This provided more sequences — and thus a larger sample — for small velocities than for large velocites.

Figure 20 displays the ratio of the calculated x, y, and z coordinates to those determined from the encoder position along the track. As the figure illustrates, the system behavior is acceptable until the velocity of the object to which the hydrophones are attached reaches 1.5 meters per second. At this point, the x and z coordinates as calculated by the monitoring computer software begin to deviate significantly from the coordinates as predicted by the encoder position. Until this velocity is achieved

the coordinates as determined by the algorithm are acceptable. The system software calculates coordinates as if the hydrophones were at the midpoint of the distance traversed during the thumper sequence. A large c_{hk} will be determined because the hydrophone/thumper ranges from the midpoint coordinates will not correspond to the ranges actually acquired by the receiver, but since the coordinates calculated match what is expected this is not necessarily bad.

However, the above analysis leads to another concern. The system is required to produce coordinates of the hydrophones at particular places in time. The preceding experiment shows that the system is calculating the coordinates of the hydrophones when they were in the middle of the thumper sequence. Thus the coordinates are calculated late — the hydrophones are no longer at the coordinates shown when those coordinates are determined. The coordinate information lags the actual positions of the hydrophones when the hydrophones are in motion. This is not necessarily critical: for vehicles using the data, a classical control system can readily deal with lagging information, as can Kalman filtering methods.

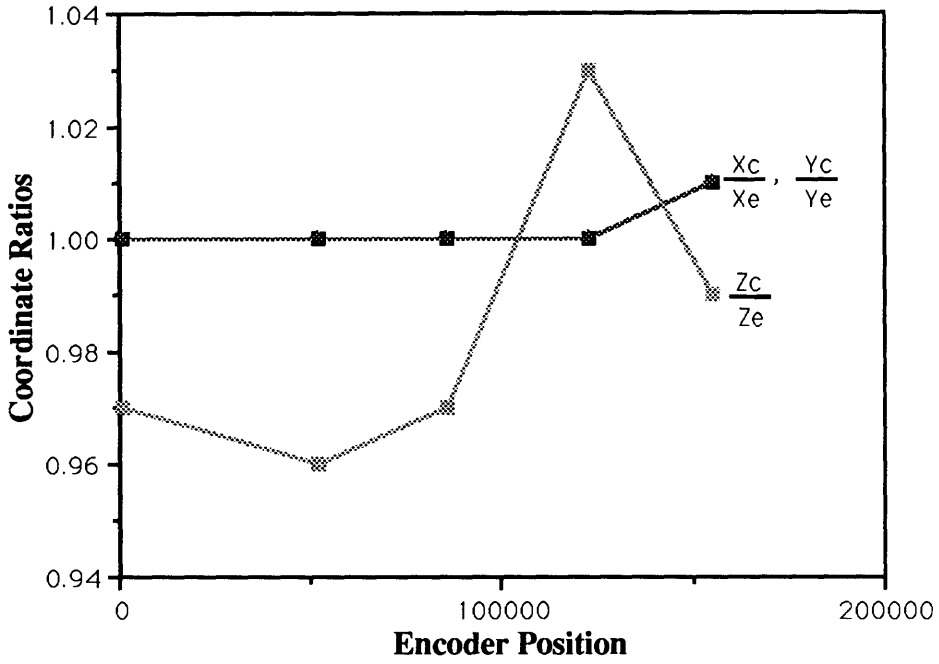


Figure 13. Ratio of Calculated Coordinates vs. Encoder Position. Hydrophone 0. Static Test.

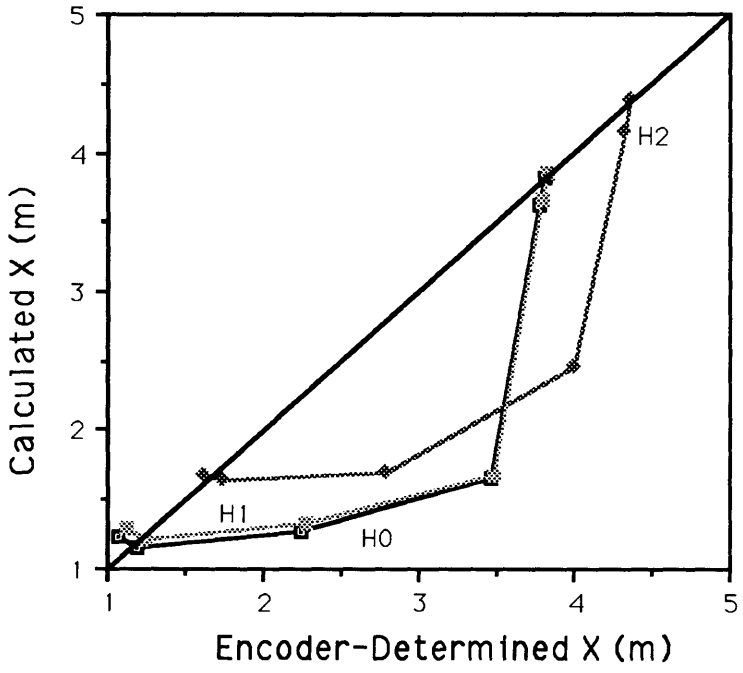


Figure 14. Calculated X versus Encoder-Determined X Dynamic Test 1

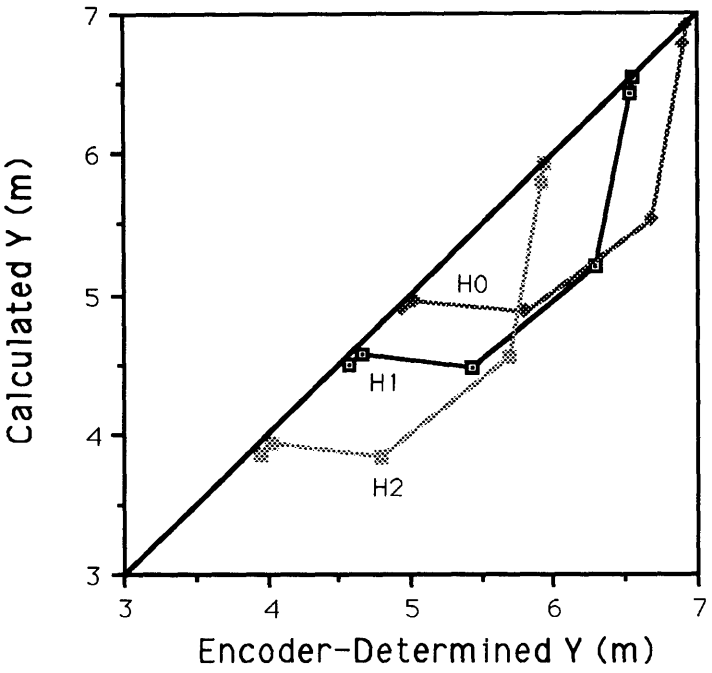


Figure 15. Calculated Y versus Encoder-Determined Y Dynamic Test 1

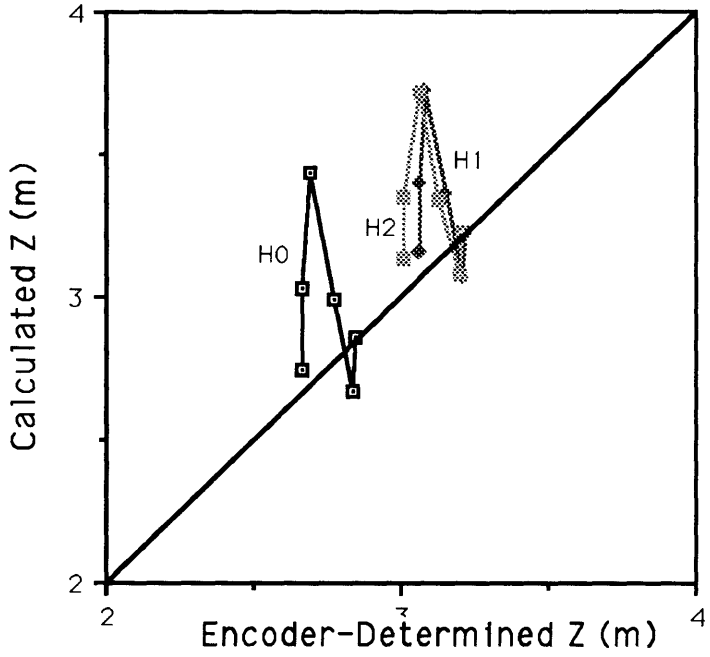


Figure 16. Calculated Z versus Encoder-Determined Z Dynamic Test 1

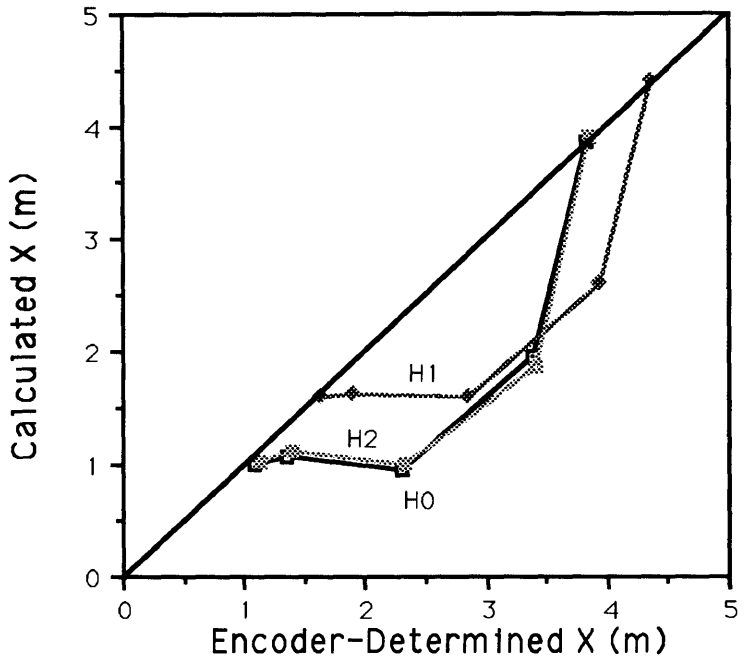


Figure 17. Calculated X versus Encoder-Determined X Dynamic Test 2

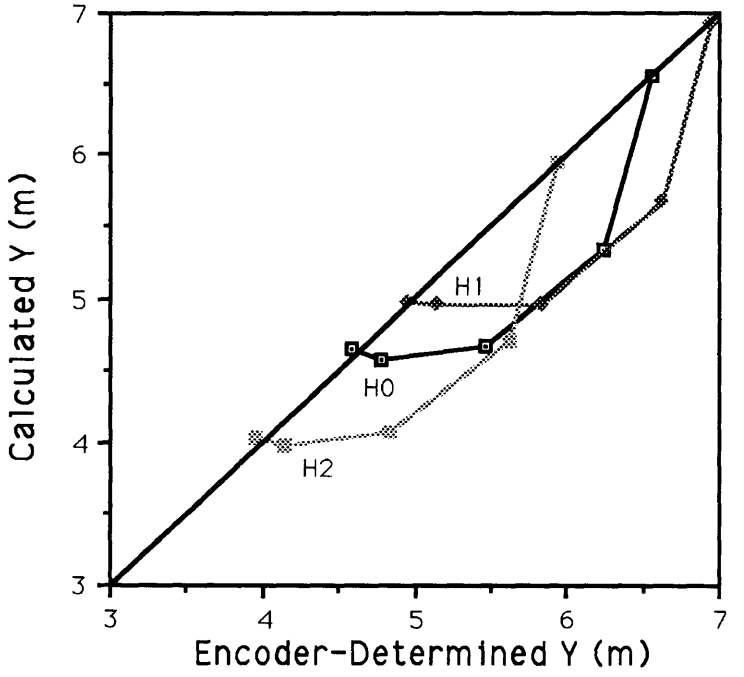


Figure 18. Calculated Y versus Encoder-Determined Y Dynamic Test 2

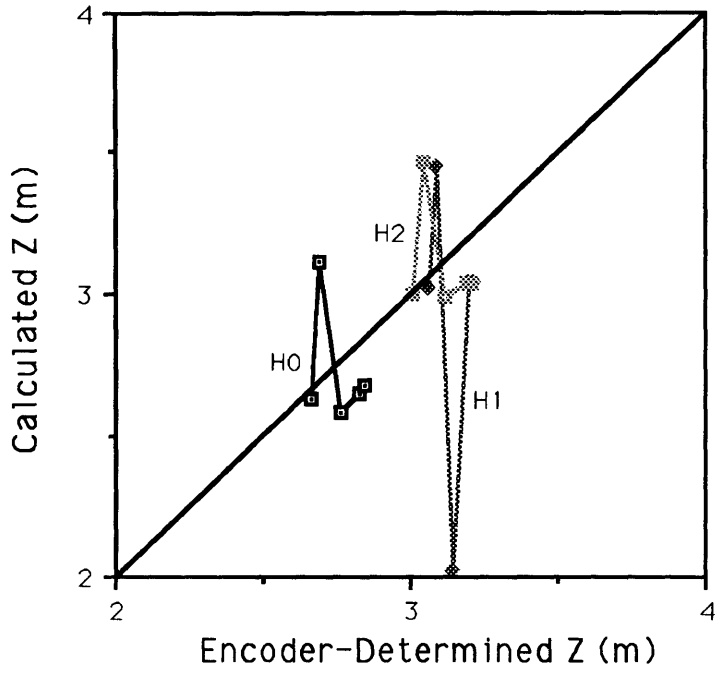


Figure 19. Calculated Z versus Encoder-Determined Z Dynamic Test 2

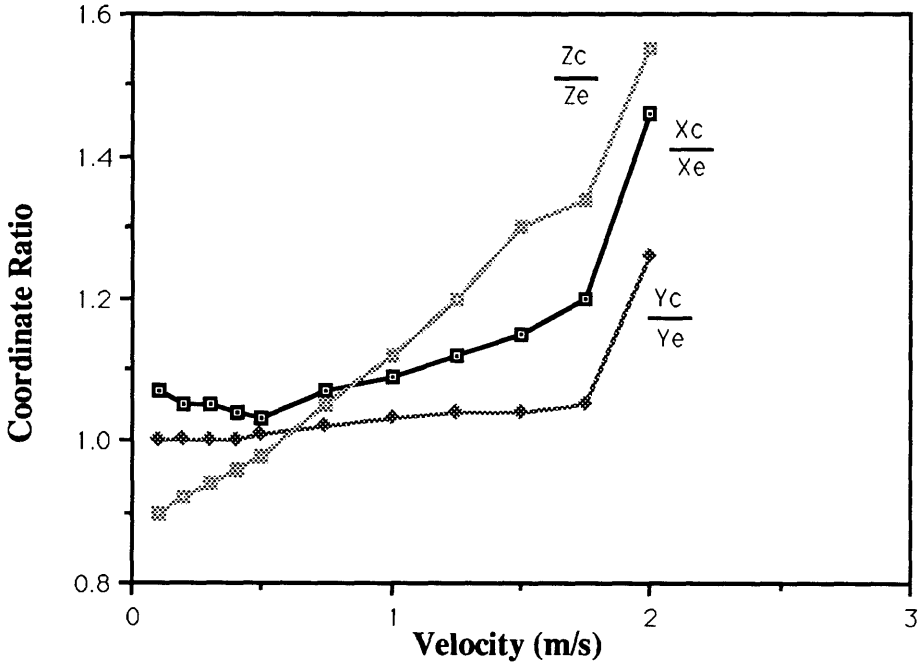


Figure 20. Velocity Error Profile

3.3 Human Reach Envelopes

In order to more fully determine the capabilities of humans in space, a series of measurements was devised to determine the effective "volume of action" allowed to a human being wearing a pressure suit. This space, called the reach envelope, illustrates the restrictions placed on a human because of the necessity of the pressure suit. By accurately determining the full reach envelope of a test subject in a pressure suit and comparing it to the reach envelope of the same test subject without a pressure suit, the limitations due to the suit can be clearly noted. Through the use of suited and unsuited reach envelopes pressure suit design may be improved.

The SSL's neutral buoyancy research allows the capability of accurate reach envelope determination. By utilizing the Three-Dimensional Acoustic Positioning System, the extent to which a human can reach, in or out of a pressure suit, can be accurately measured. Because of the neutrally buoyant environment, the reach envelope data gathered is directly applicable to in-orbit operations. This is an improvement over NASA's method of determining reach envelopes which measures only upper body maneuvers in a laboratory environment.

The reach envelope tests were conducted at both the MIT pool and the Marshall tank. In the MIT pool, three subjects performed unsuited reach envelope tests. Each test subject stood in foot restraints placed in the test area, which were attached to the IBS stand as the hydrophone cluster in the static tests had been. The test subjects were supplied with air from a scuba bottle placed on the bottom, so that their movements would not be constrained by wearing scuba tanks and a harness. The PVC hydrophone cluster was made neutrally buoyant with the aid of floatation elements to reduce the amount of force required to maneuver it. An aluminum rod was attached to the base of the PVC cluster for use as a handle. Each test subject performed maneuvers to the full extent allowed by the test setup. Holding the cluster by the rod, the subjects maneuvered in an attempt to extend the cluster to the limits

of their reach.

The reach envelope tests conducted at the Marshall tank were performed on nine test subjects, all wearing pressure suits. The three subjects tested at the MIT pool were among those tested at the Marshall tank. The setup was similar: test subjects were placed in foot restraints within the test area, and, while in a pressure suit, performed maneuvers to extend the hydrophone cluster to the limits of their reach. In addition, tests were conducted to determine reach envelopes with only one foot in the restraint. For tests at both the MIT pool and the Marshall tank, a static test was conducted to determine the coordinates, within the respective coordinate systems, of the foot restraints in which the subjects stood. These coordinates were:

$$(x, y, z) = (5.9, 6.3, -2.4) \text{ for MIT pool}$$

$$(x, y, z) = (5.8, 0.0, 4.7) \text{ for Marshall tank}$$

For all subjects, the range data was converted to coordinate and attitude data for each hydrophone. Then, using a simple transformation, equation (3.3-1), the origin of the coordinate system assigned to the PVC cluster was calculated, averaging the origins as determined from each hydrophone. This coordinate was defined as the limit to which each of the test subjects could reach.

$$\begin{aligned} x_c &= x_p - t_{11}\bar{x}_p - t_{12}\bar{y}_p - t_{13}\bar{z}_p \\ y_c &= y_p - t_{21}\bar{x}_p - t_{22}\bar{y}_p - t_{23}\bar{z}_p \\ z_c &= z_p - t_{31}\bar{x}_p - t_{32}\bar{y}_p - t_{33}\bar{z}_p \end{aligned} \qquad (3.3-1)$$

where

$(x, y, z)_c$ = Cluster Origin in Inertial System Coordinates
$(x, y, z)_p$ = Hydrophone Location in Inertial System Coordinates
$(\bar{x}, \bar{y}, \bar{z})_p$ = Hydrophone Location in Cluster Coordinates

and

$$T = \begin{pmatrix} t_{11} & t_{12} & t_{13} \\ t_{21} & t_{22} & t_{23} \\ t_{31} & t_{32} & t_{33} \end{pmatrix}$$

$$T = \begin{pmatrix} \cos\psi\cos\theta & \sin\psi\cos\phi - \cos\psi\sin\theta\sin\phi & -\cos\psi\sin\theta\cos\phi - \sin\psi\sin\phi \\ -\sin\psi\cos\theta & \sin\psi\sin\theta\sin\phi + \cos\psi\cos\phi & \sin\psi\sin\theta\cos\phi - \cos\psi\sin\phi \\ \sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{pmatrix}$$

where θ, ϕ, and ψ are the three Euler angles of the PVC cluster. These angles were calculated as detailed in section 2.3. The locations of the hydrophones in the coordinate system assigned to the PVC cluster are listed in appendix C, table C-1.

The data recorded for each test was mapped to a coordinate system with its origin at the coordinates of the foot restraints at the facility used. This converted the determined reach envelopes to reach envelopes relative to the subject being tested. The coordinate system was defined with the z-axis positive up, the x-axis positive right, and the y-axis positive forward, with respect to the subject in the restraints.

The test subjects who performed both suited and unsuited maneuvers (subjects one, two, and three) demonstrate the limitations of human reach envelopes due to pressure suit restrictions. Figures 21 through 23 show the combined suited and unsuited reach envelopes for the three subjects. The graphs are two-dimensional views of the data points, viewed from all three axes. Most apparent is that while in the pressure suit the subjects were significantly prohibited from performing "below the belt" reaches, in both the x and y axes, whereas the subjects had considerably more freedom to perform such maneuvers while unsuited. This is the greatest limitation visible in the suited versus unsuited envelope comparison.

One other apparent limitation is a reduction in overall reach, aside from the above limitation. In the comparison plots of these three subjects the suited reach envelopes in the x and y axes are restricted by approximately 10% to 20% of the unsuited reach envelopes. However, the upper z-axis reaches of the three subjects appear to be unaffected by the presence or absence of the pressure suit. These limitations indicate

that the pressure suit design above the waist does not greatly limit the movement or reach of a human being wearing the suit, but that maneuvers reaching to locations below the waist, when the human must bend down, are significantly restricted.

The next four figures, 30 through 33, display the differences between single-hand and two-hand maneuvers, for cases in which both feet are in restraints and in which only one foot is restrained. Figures 30 and 31 show single-hand and two-hand maneuvers with both feet restrained. The subject in these tests demonstrated that an individual in a pressure suit has a greater reach with either single hand than with both hands together. Using the maximum x, y, and z values achieved, the following results are evident:

$$\max \Delta x_{1H} = 3.3 \text{ m}$$
$$\max \Delta x_{2H} = 1.6 \text{ m}$$
$$\max \Delta y_{1H} = 1.4 \text{ m}$$
$$\max \Delta y_{2H} = 0.7 \text{ m}$$
$$\max \Delta z_{1H} = 2.0 \text{ m}$$
$$\max \Delta z_{2H} = 1.7 \text{ m}$$

It can be seen that the x and y reach envelopes for two-hand maneuvers are 50% of those for single-hand maneuvers. The z reach envelope for two-hand maneuvers is 85% of that for single-hand maneuvers, indicating that the reach of an individual in the z-axis is not greatly affected by using one or both hands. Also, the figures show that the z envelope upper limit is extended from the two-hand test to the single-hand test, while the lower limit remains approximately the same. This was not true for the x and y reach envelopes: the limits were extended in both directions.

Figures 32 and 33 show the results of single- and two-hand maneuvers while only one foot was restrained. This allowed the subject greater flexibility and consequently a greater reach envelope. Figure 32 displays the single-hand test while figure 33 shows the two-hand maneuvers. Again, it is apparent that the relationships between single- and two-hand maneuvers for the both-feet restrained case hold true for the one-foot

restrained case: the x and y reach envelopes differ by a factor of two from single- to two-hand maneuvers while the z reach envelope varies by the same amount as the both-feet restrained case. However, the x and y reach envelopes are larger for the one-foot restrained case than the both-feet restrained case. The y reach was extended by a factor of 1.6 in the single-hand test and a factor of 2.6 in the two-hand test, from the both-feet restrained case to the one-foot restrained case. The x reach was extended by a factor of 0.15 in the single-hand test and a factor of 0.32 in the two-hand test. The z reach envelope was relatively unaffected, but is shifted downward along the z-axis because the test subject had more freedom to perform "below the belt" maneuvers. As in the both-feet restrained case, the x and y reach envelopes are extended in both directions from the two-hand to the single-hand maneuvers. The z reach envelope is also extended in both directions, but to a significantly smaller degree.

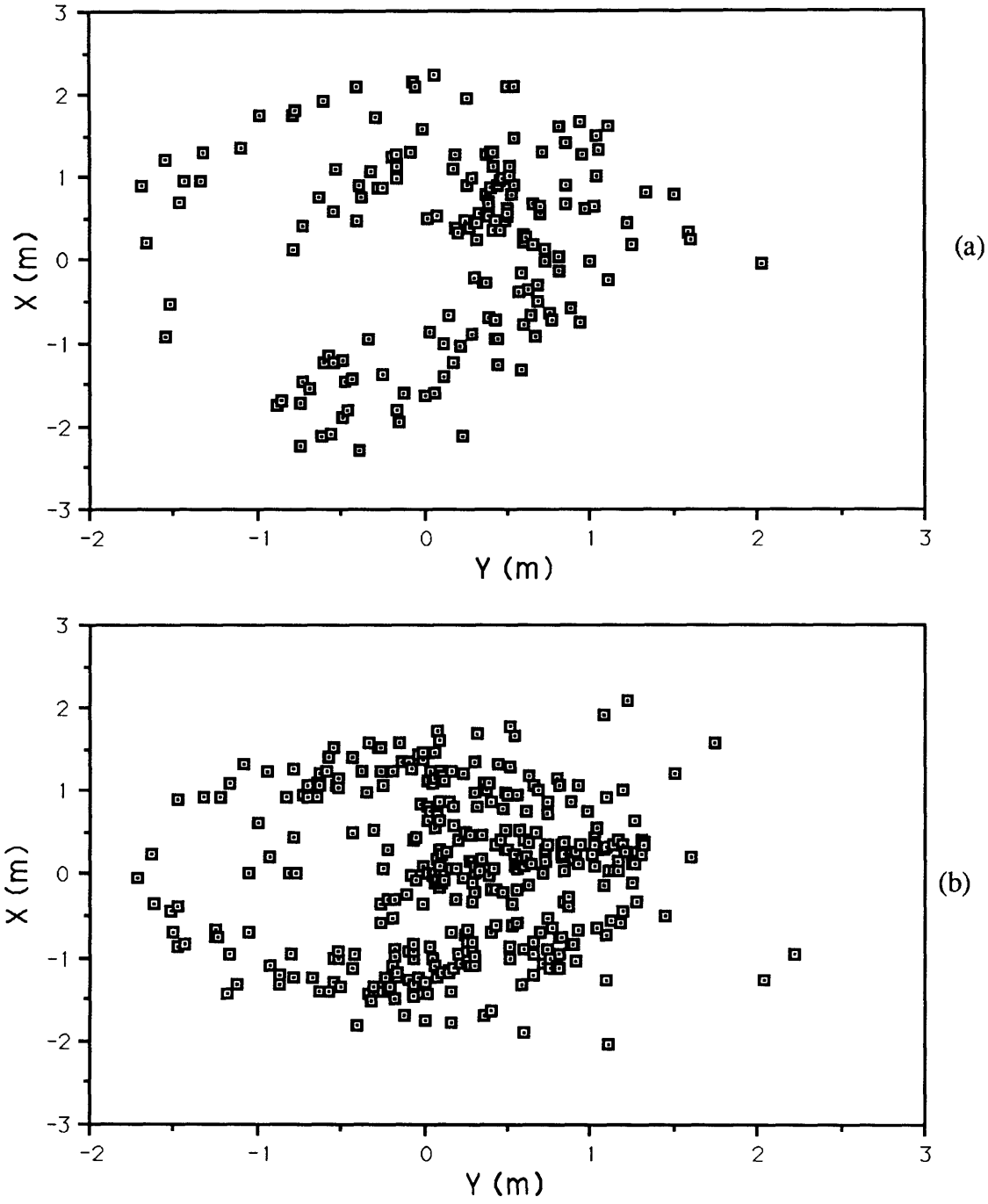


Figure 21. Subject 1 Reach Envelope
(a) Unsuited (b) Suited

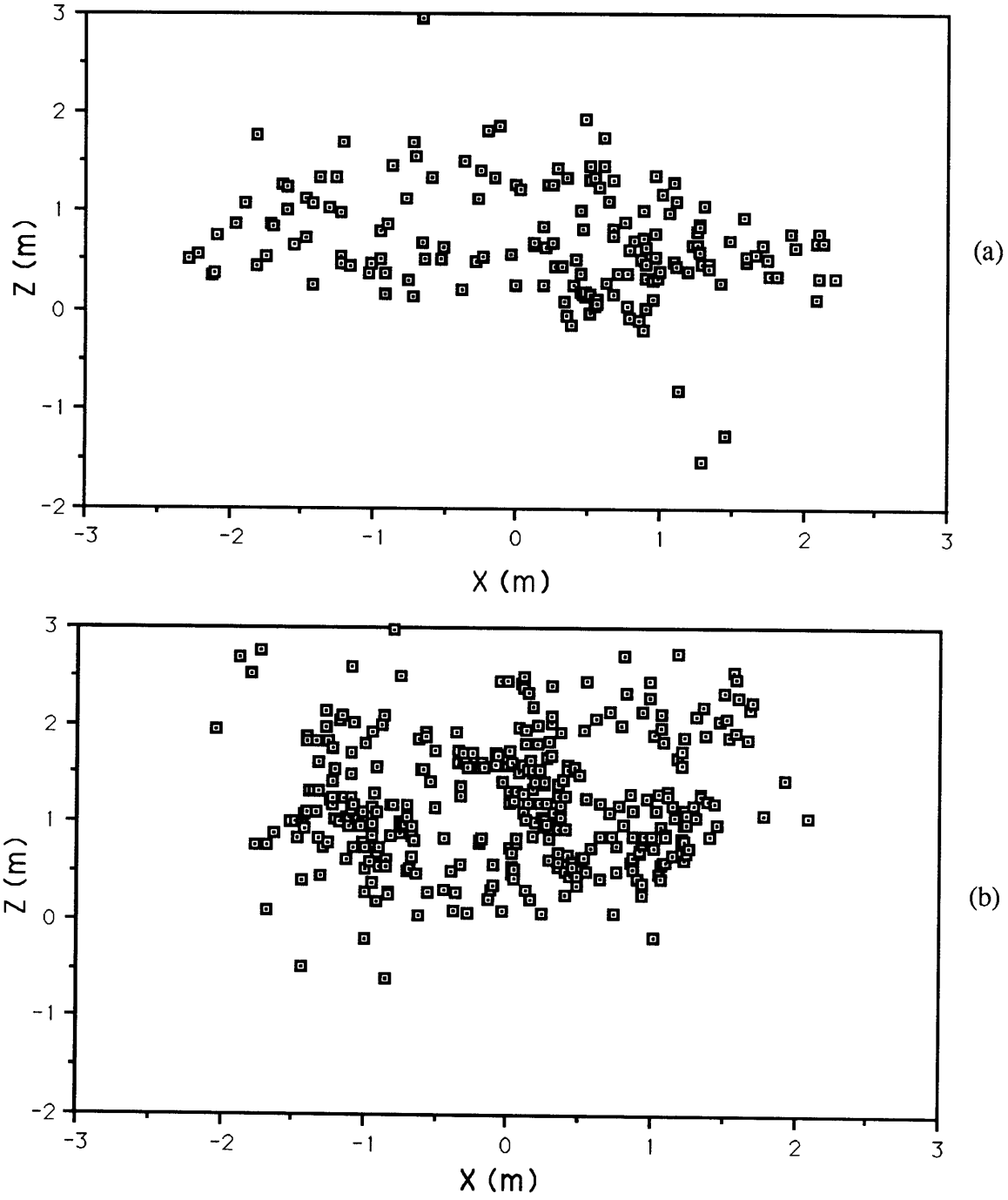
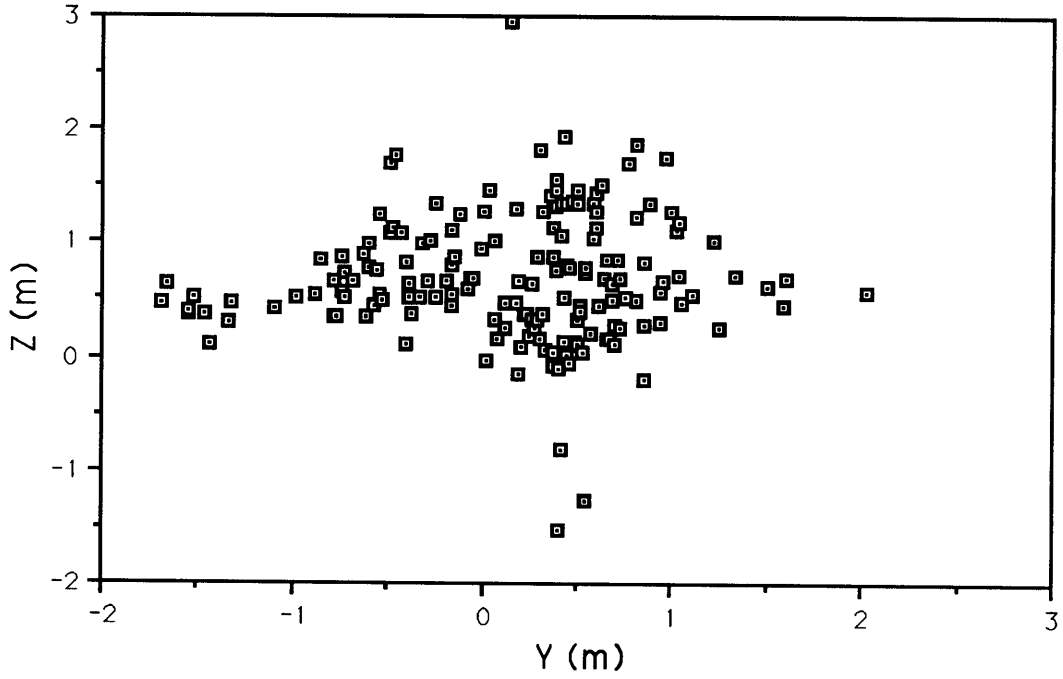
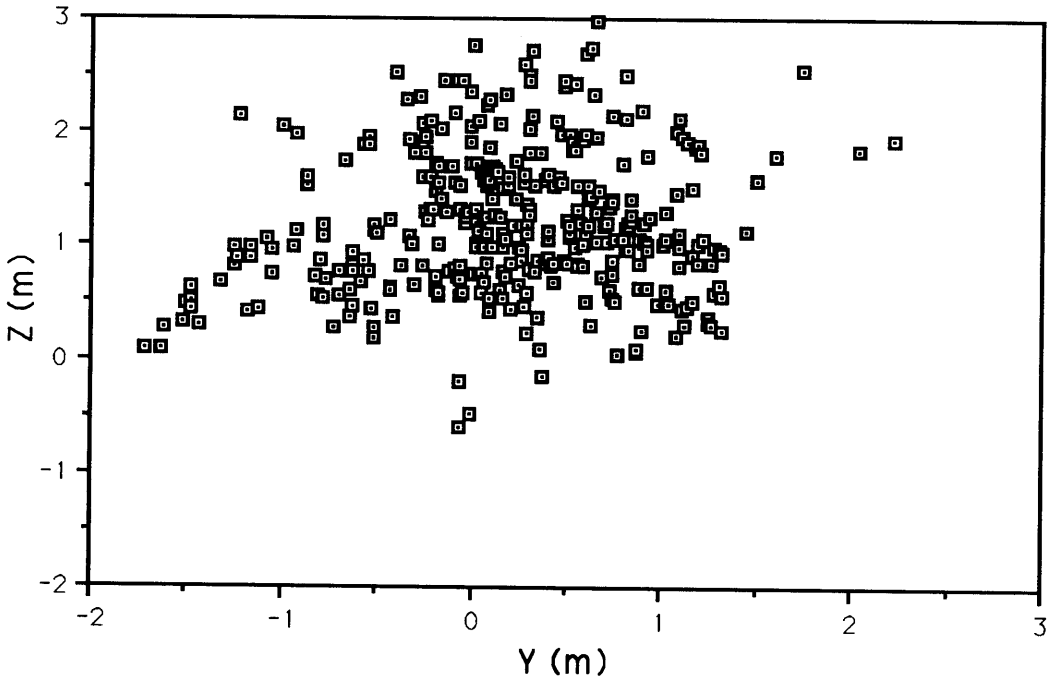


Figure 22. Subject 1 Reach Envelope:
(a) Unsuited (b) Suited



(a)



(b)

Figure 23. Subject 1 Reach Envelope:
(a) Unsuited (b) Suited

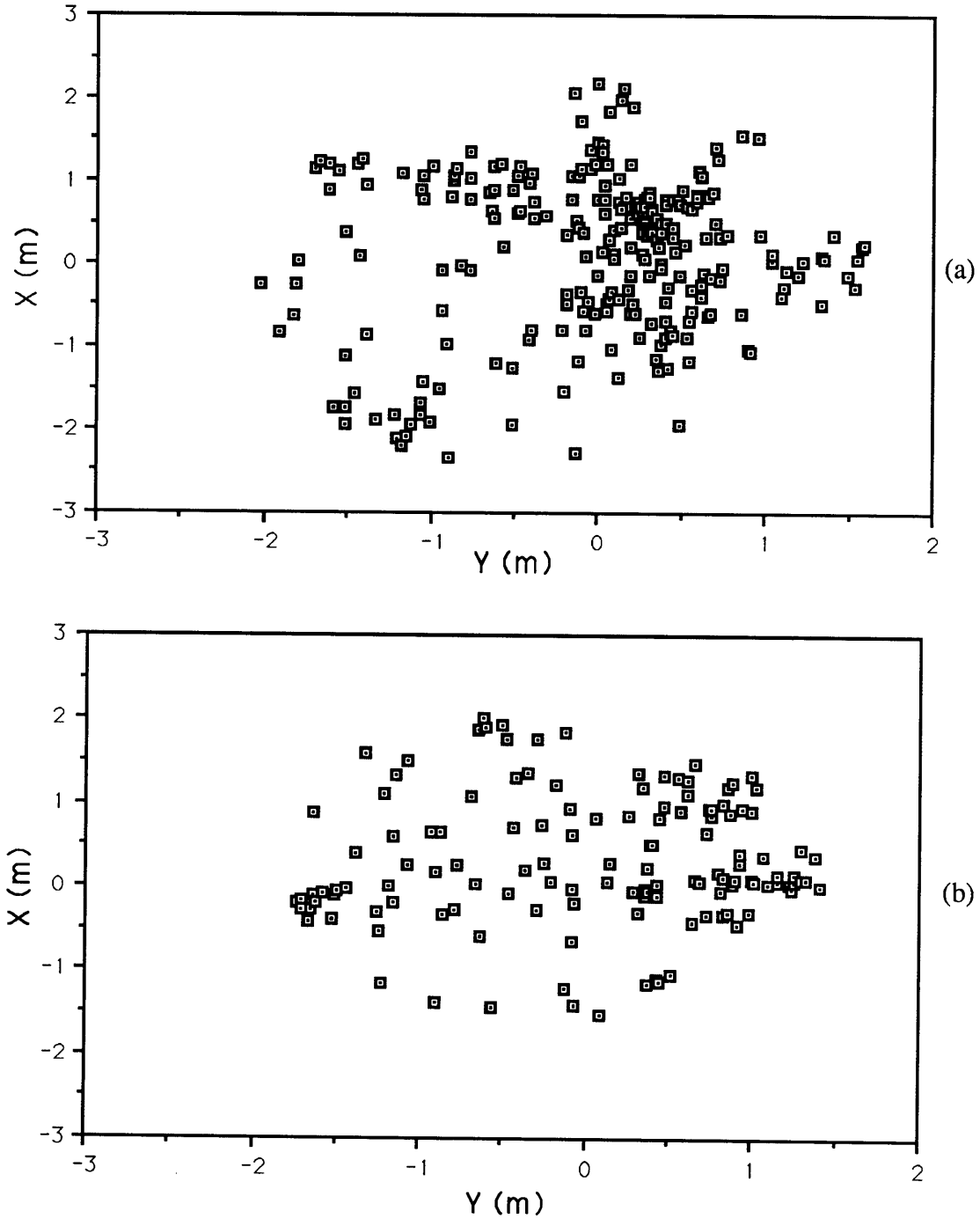


Figure 24. Subject 2 Reach Envelope:
(a) Unsuited (b) Suited

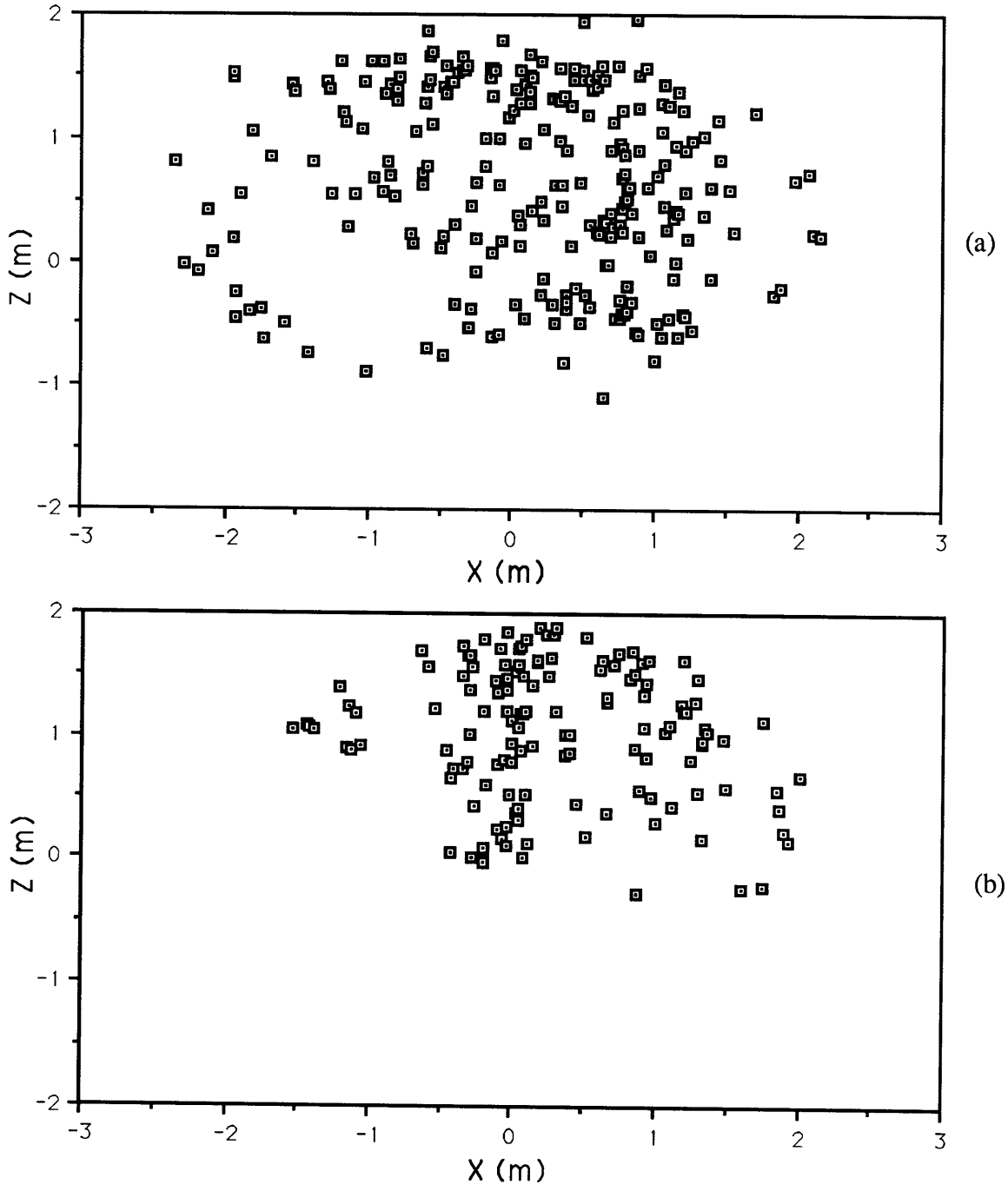


Figure 25. Subject 2 Reach Envelope:
(a) Unsuited (b) Suited

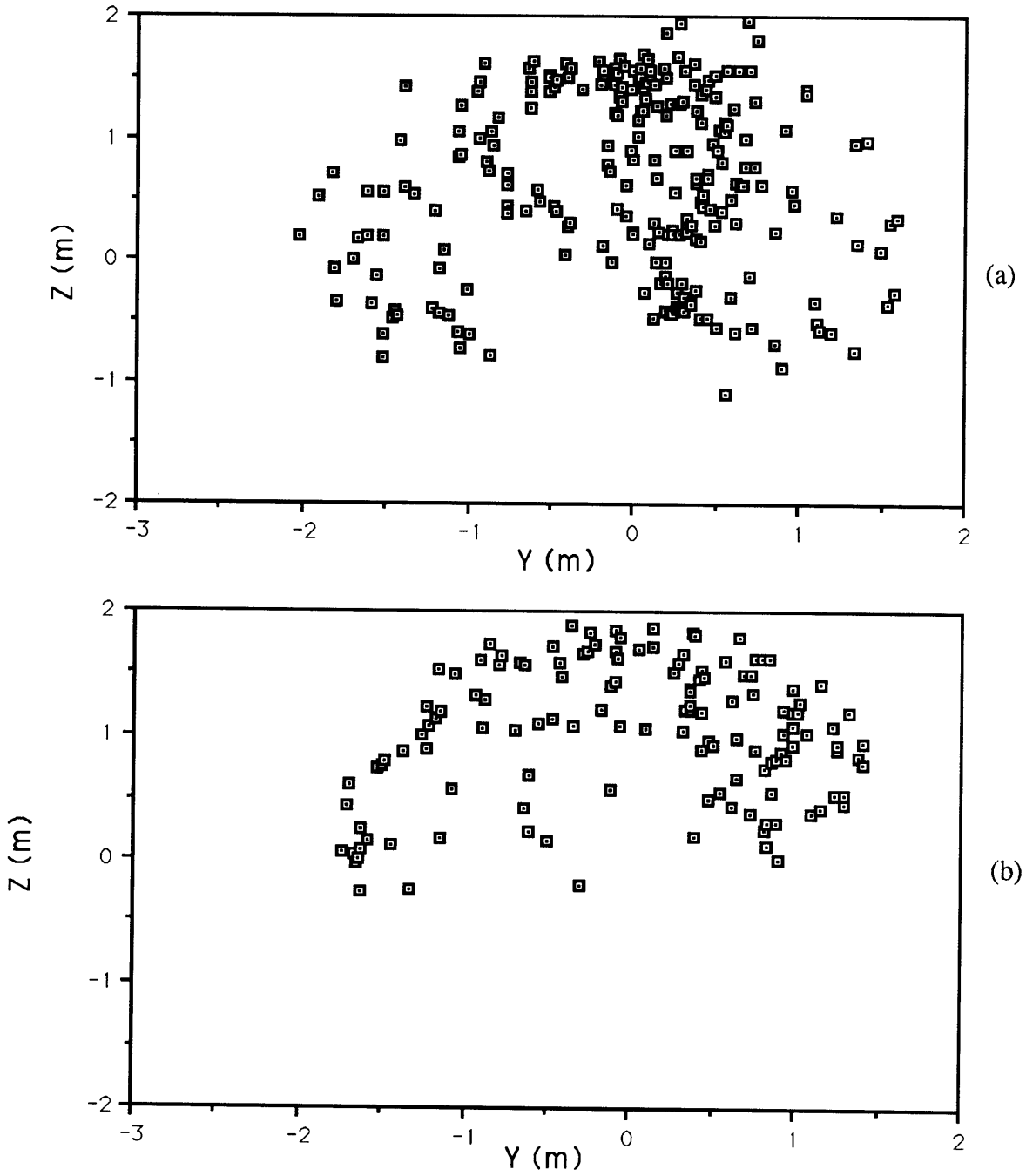


Figure 26. Subject 2 Reach Envelope:
(a) Unsuited (b) Suited

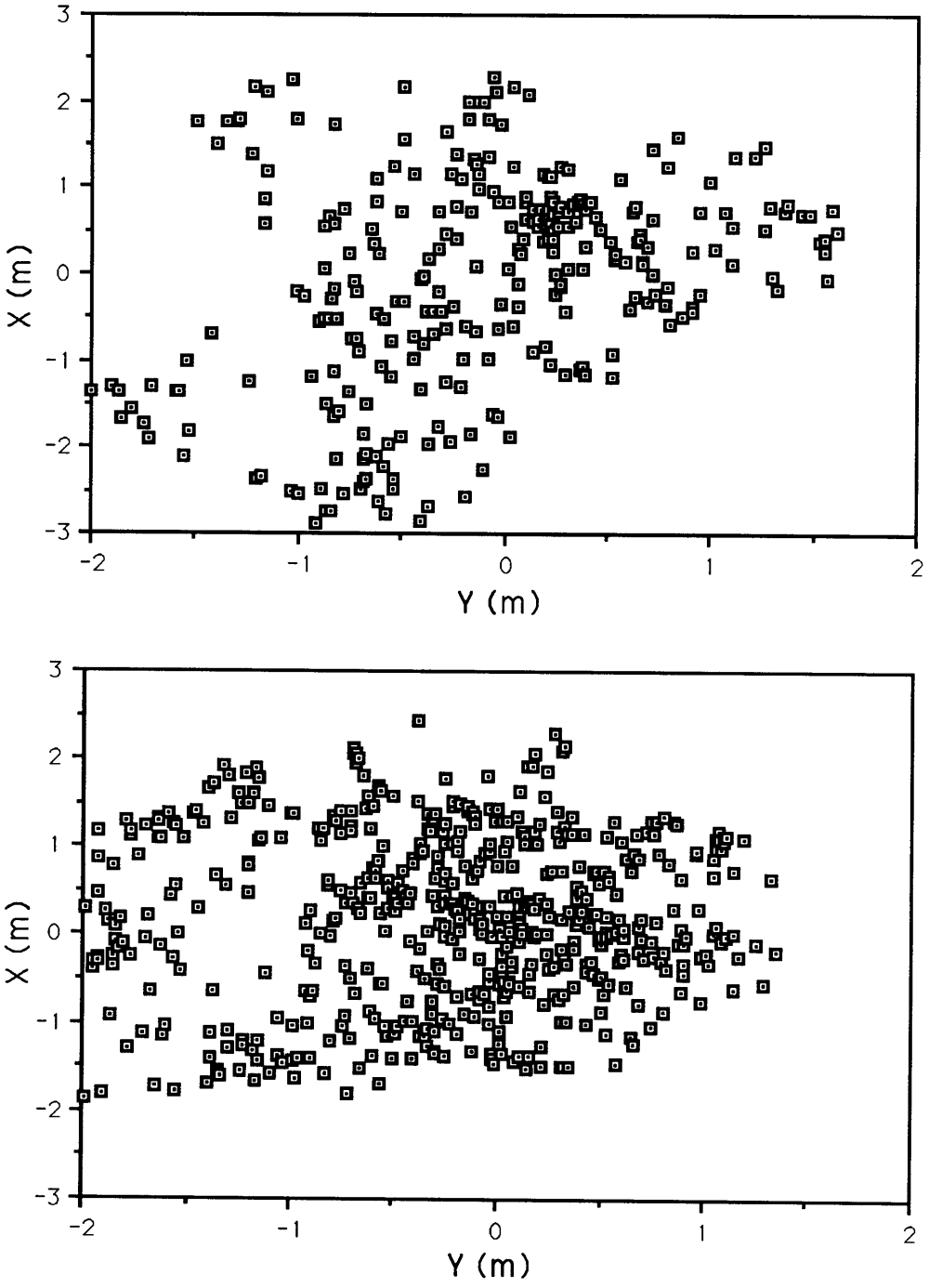


Figure 27. Subject 3 Reach Envelope:
(a) Unsuited (b) Suited

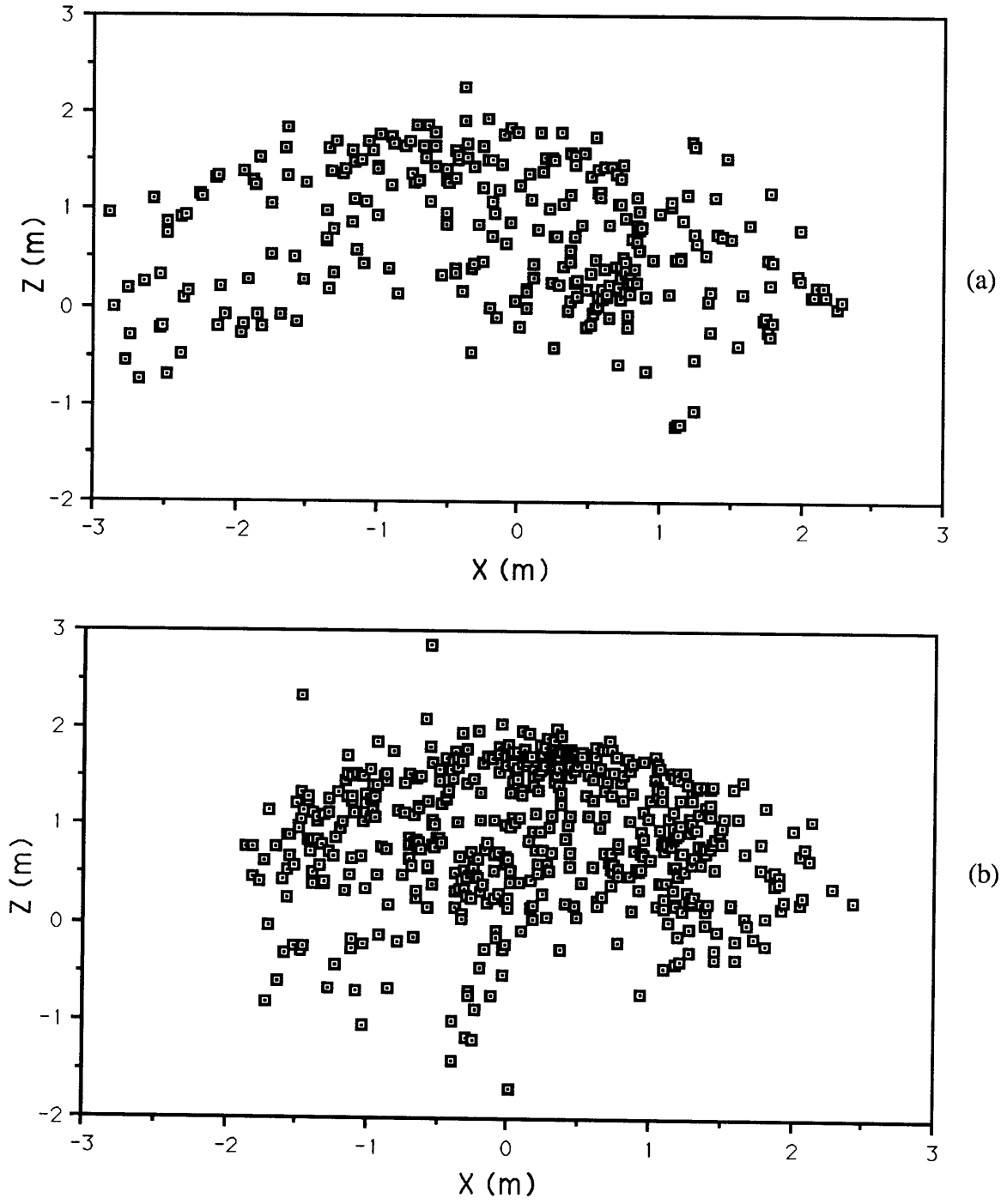
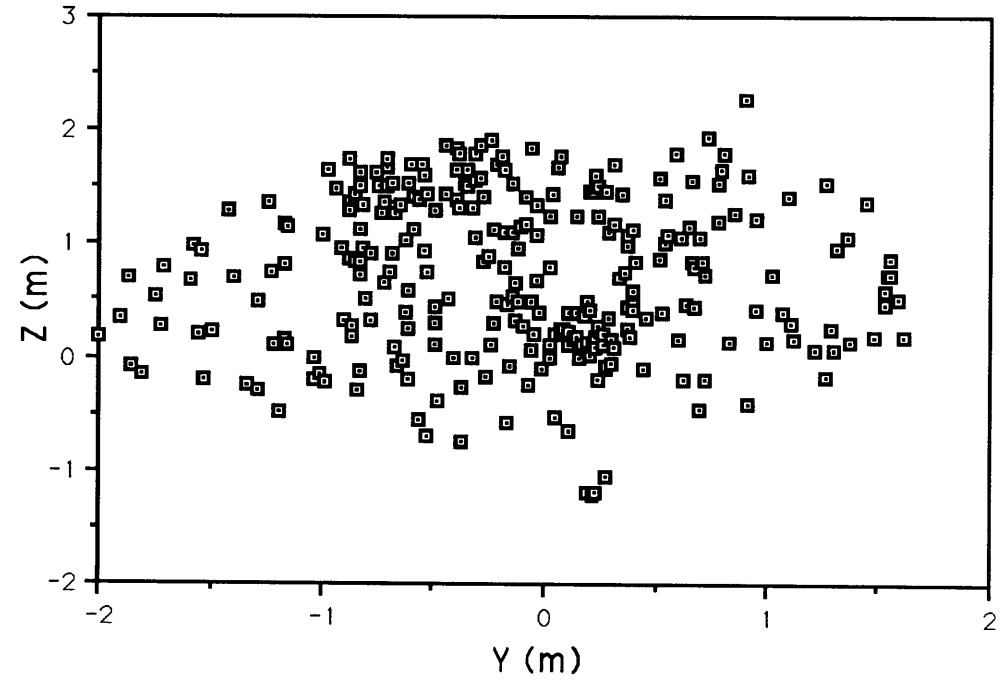
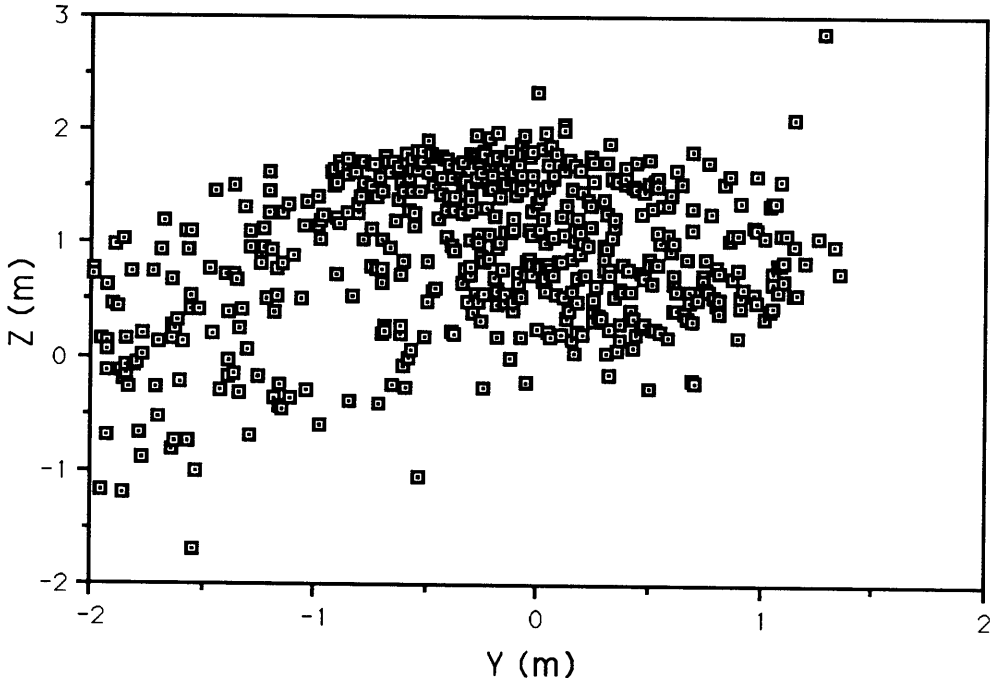


Figure 28. Subject 3 Reach Envelope:
(a) Unsuited (b) Suited



(a)



(b)

Figure 29. Subject 3 Reach Envelope:
(a) Unsuited (b) Suited

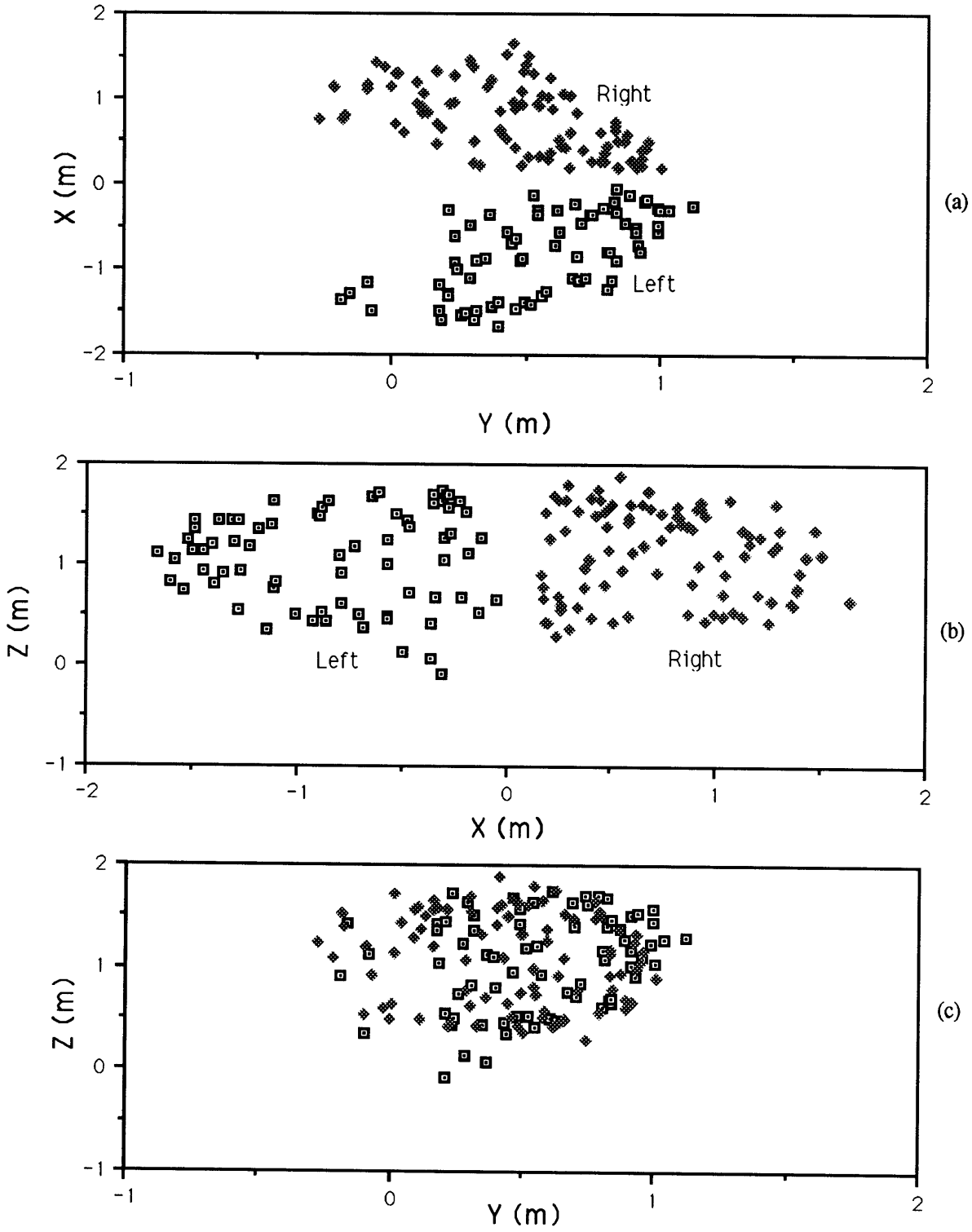


Figure 30. Subject 4 Reach Envelope:
Single-Hand Both Feet Restrained

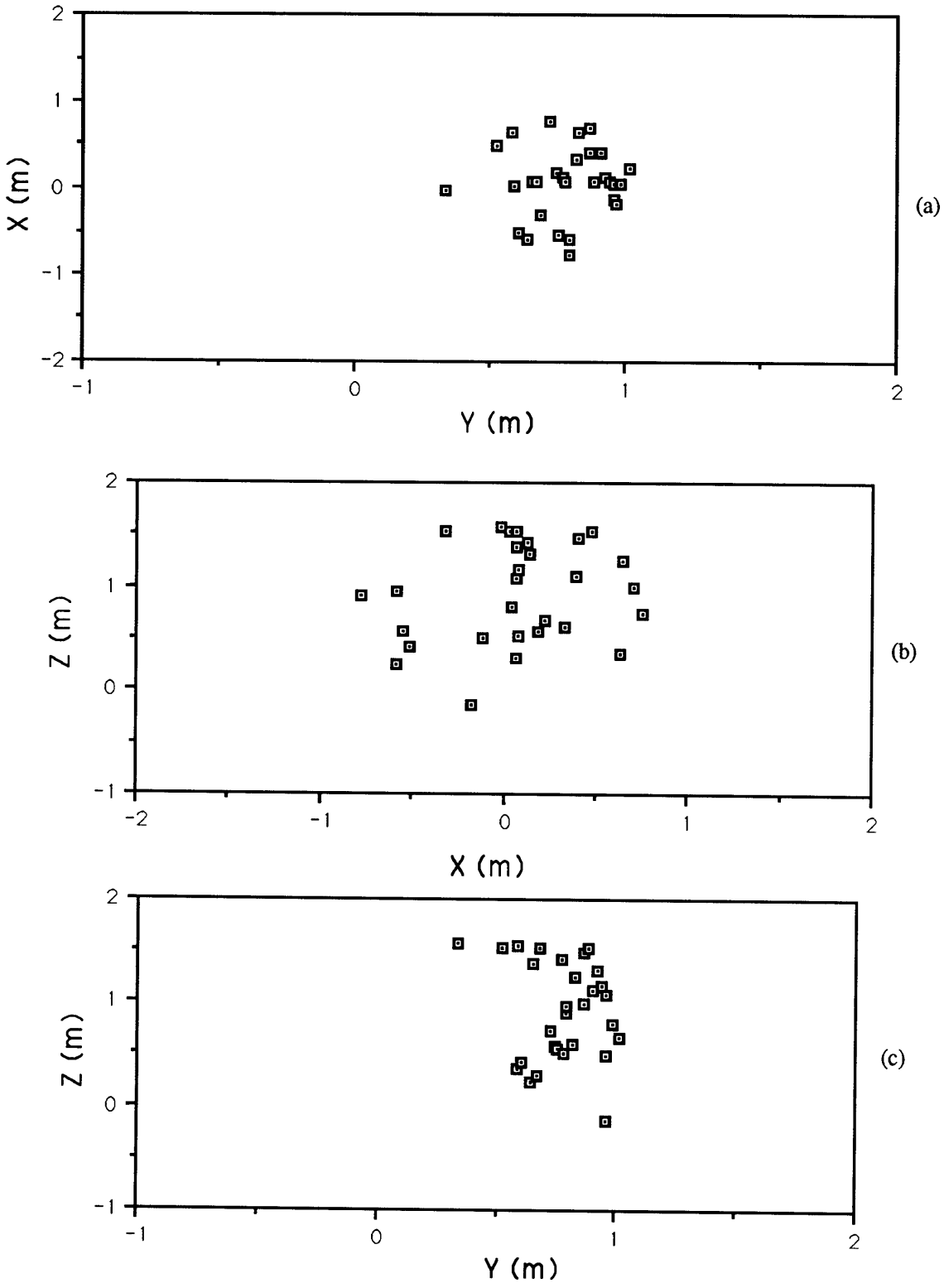


Figure 31. Subject 4 Reach Envelope:
Two-Hand, Both Feet Restrained

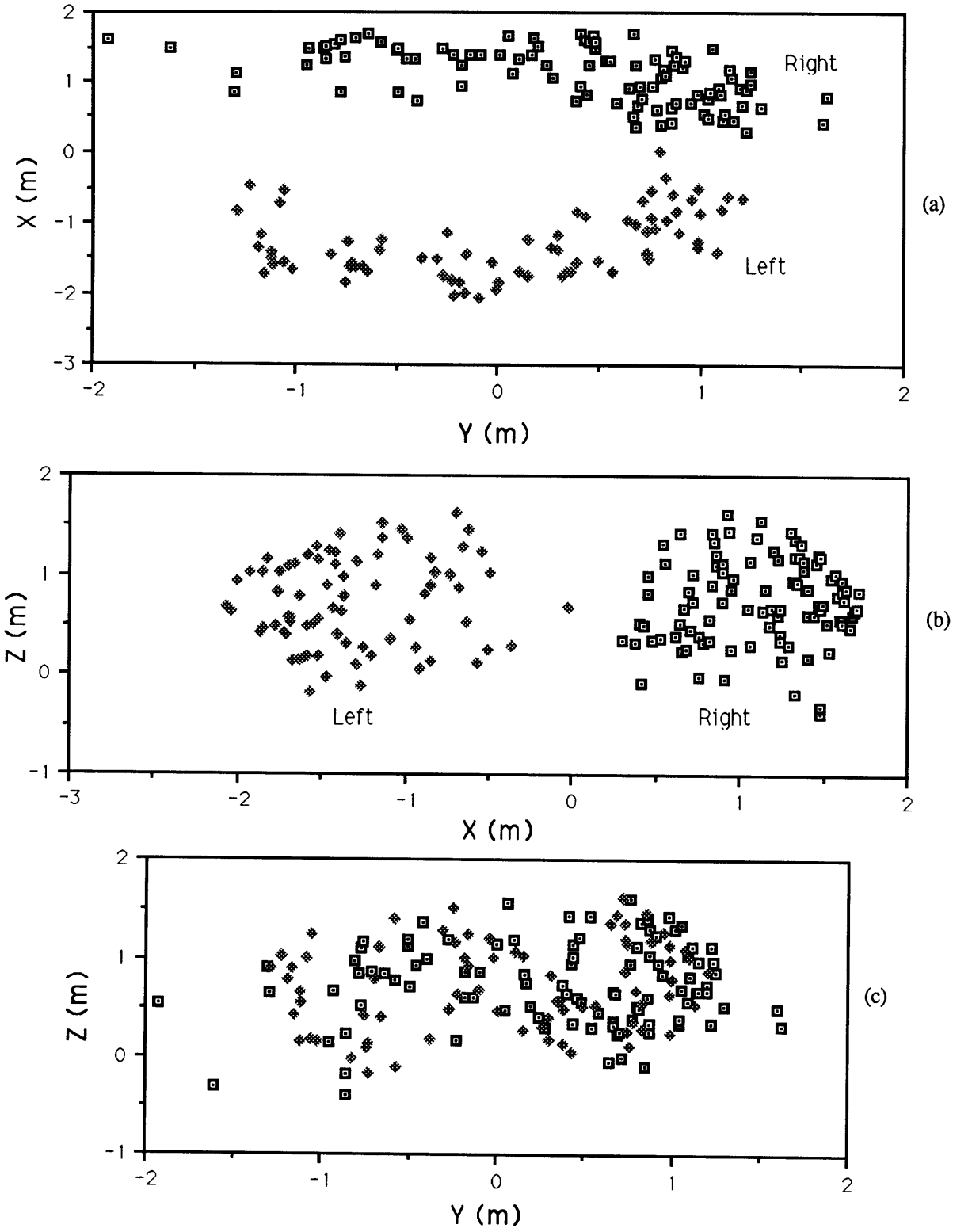


Figure 32. Subject 4 Reach Envelope:
Single-Hand One Foot Restrained

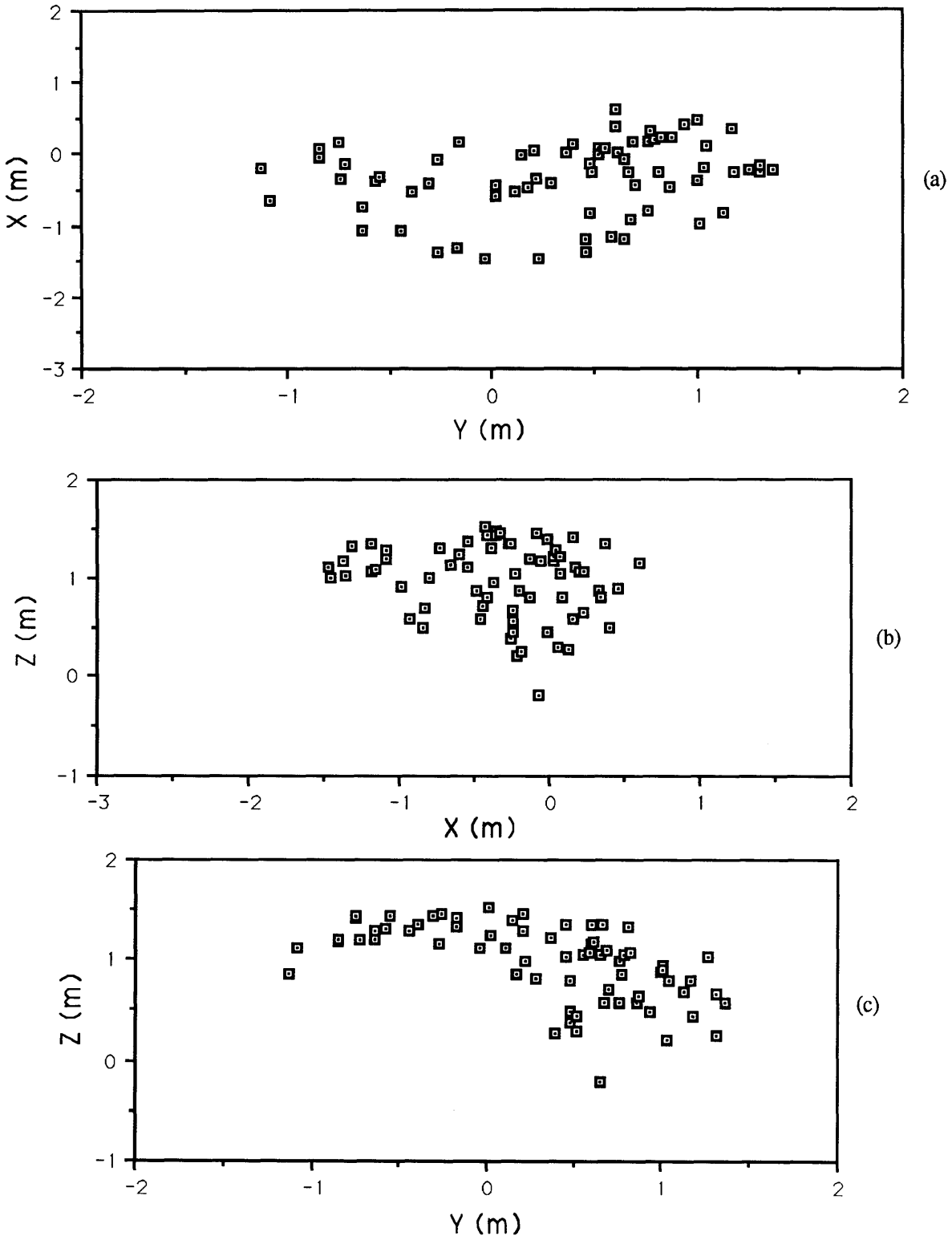


Figure 33. Subject 4 Reach Envelope:
Two-Hand One Foot Restrained

3.4 Teleoperated Robot Trajectories

The 3DAPS system was attached in a number of experiments to both the Beam Assembly Teleoperator (BAT, reference 6), and the Multimode Proximity Operations Device (reference 7). Each vehicle was controlled by an operator on the surface, and the 3DAPS system simply provided range information using the IBM to store the data until analysis could be performed later. The positions of the hydrophones were measured for the tests with respect to the coordinate system assigned to each vehicle. Appendix C contains the hydrophone location data for both BAT and MPOD. Figure 34 through 39 show trajectories of MPOD and BAT as each performs maneuvers in the Marshall Tank. The plots for each figure are discontinuous, with each discontinuity representing the 3DAPS-calculated data point while the vehicle was in motion. Tasks performed were those required for other (unrelated) test objectives, and were not chosen for 3DAPS correlation.

In figures 34 through 36 MPOD locations are displayed while performing docking maneuvers. Figure 36 shows that MPOD spent a large portion of its time in the tank maneuvering around $y = 0$, $z = 6$; this is where the docking target was located. Figure 34 shows that MPOD made a number of docking attempts parallel to the x-axis at $y = 0$. Research has been performed at the SSL using the 3DAPS system and MPOD to determine effects of different methods of information display for the MPOD operator. This research is documented in reference 8. Future plans for MPOD include adapting the 3DAPS receiver and monitoring computer to reside onboard and provide position and attitude feedback in real time for six degree-of-freedom closed-loop control (reference 9).

BAT is depicted in figures 37 through 39 while performing structure-grappling maneuvers in the Marshall tank. In the area around the point (12.5, -2, 4), BAT is attached to the Remote Manipulator System (RMS) and is maneuvered around the structure with which it is attempting to grapple. This presents another possible

application of the 3DAPS system: at present, the Marshall tank RMS has no position or attitude feedback for the end effector at the end of its arm. The 3DAPS system can readily be employed to determine the trajectory of the RMS as it maneuvers within the Marshall tank. Such research would provide meaningful information regarding the behavior of the arm in response to its controller.

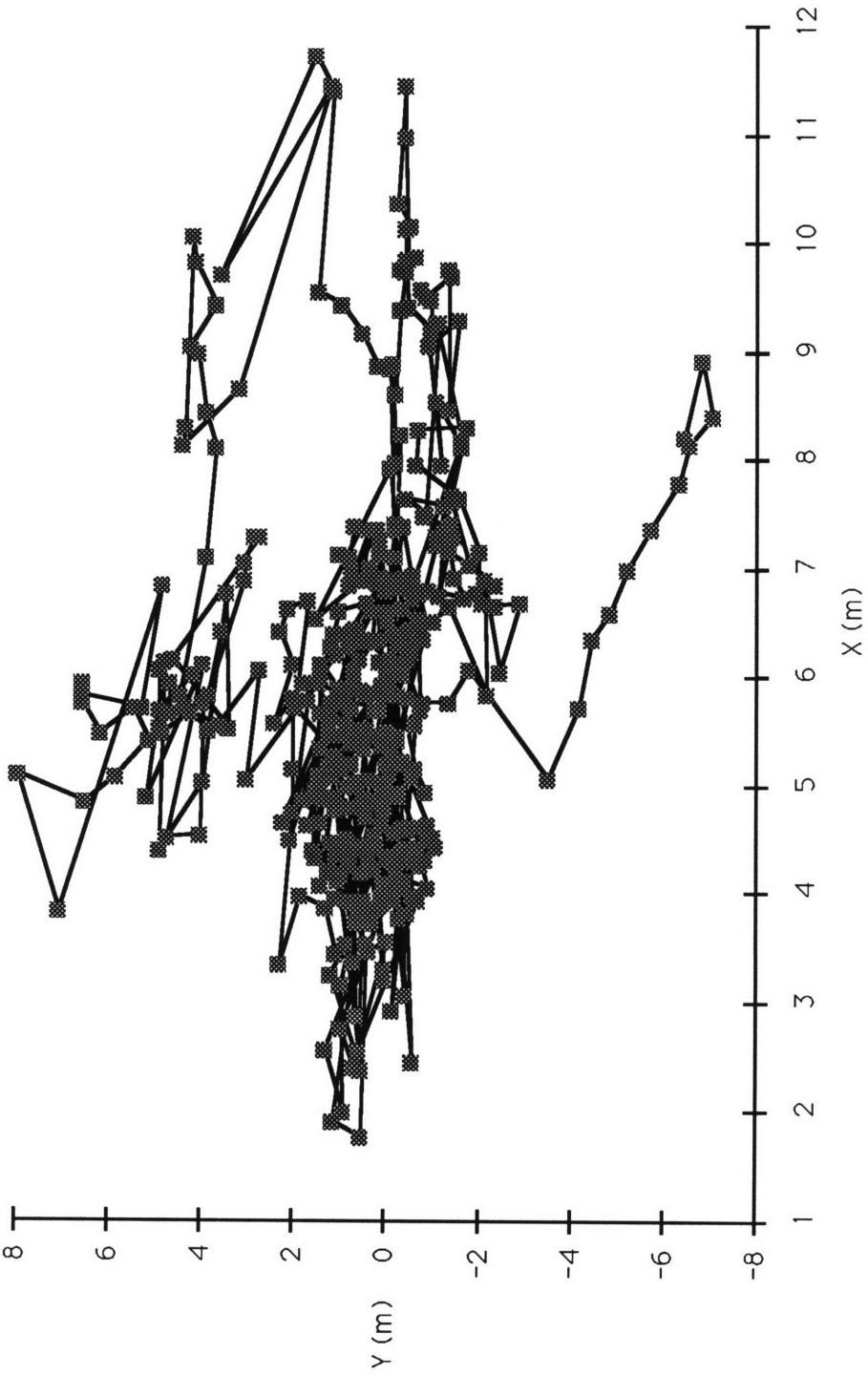


Figure 34. MPOD Performing Docking Maneuver at Marshall Tank

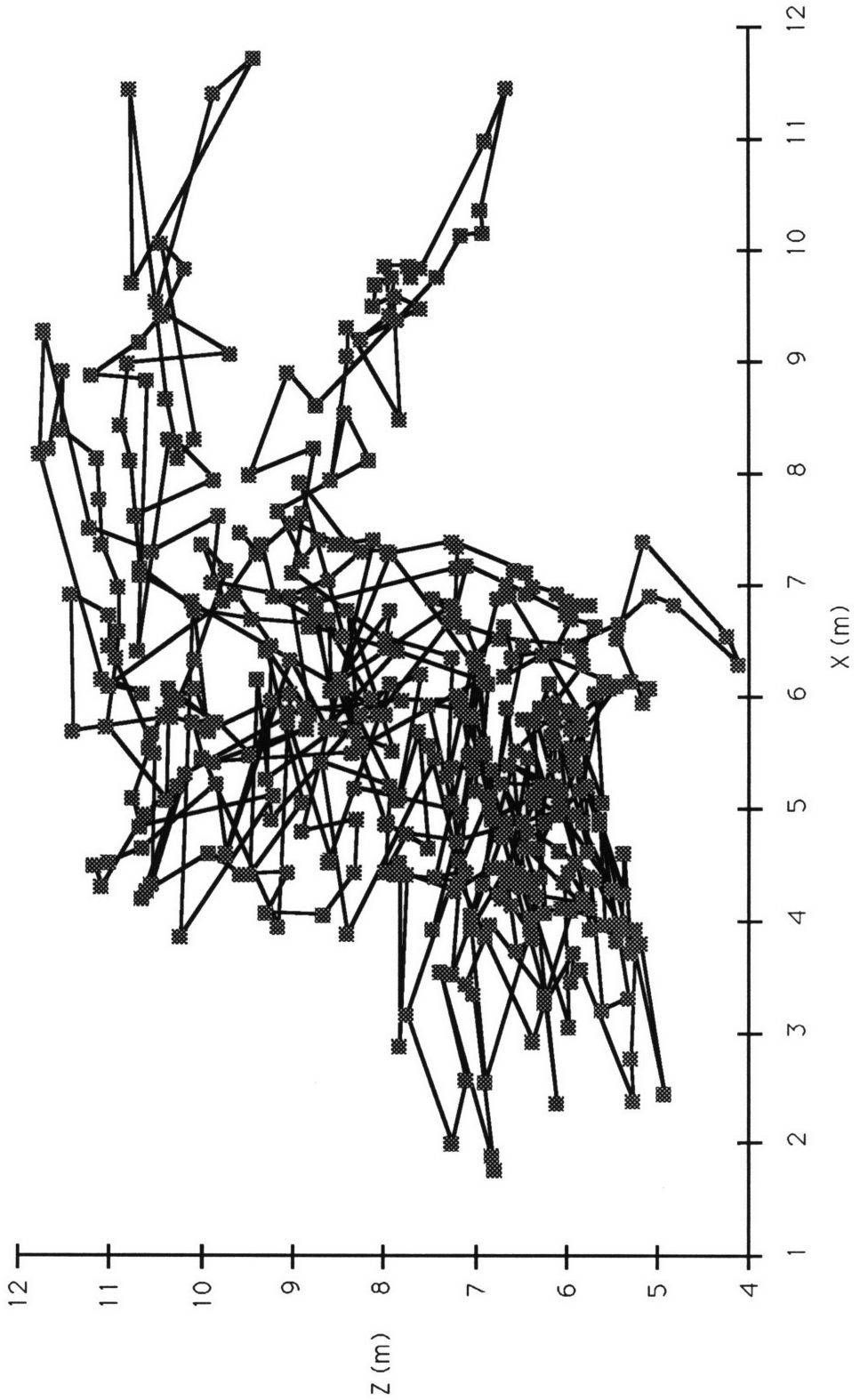


Figure 35. MPOD Performing Docking Maneuvers in Marshall Tank

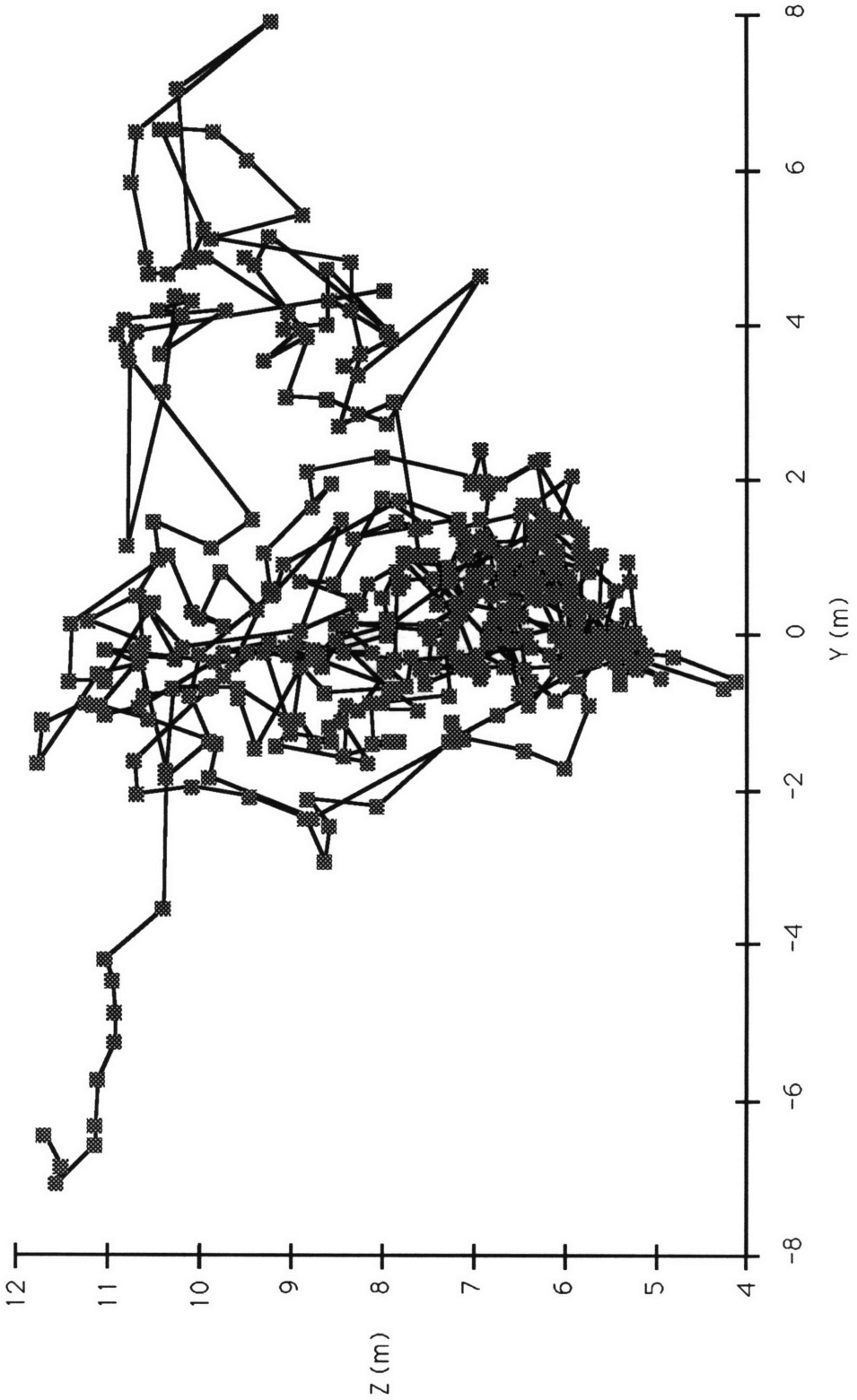


Figure 36. MPOD Performing Docking Maneuvers in Marshall Tank

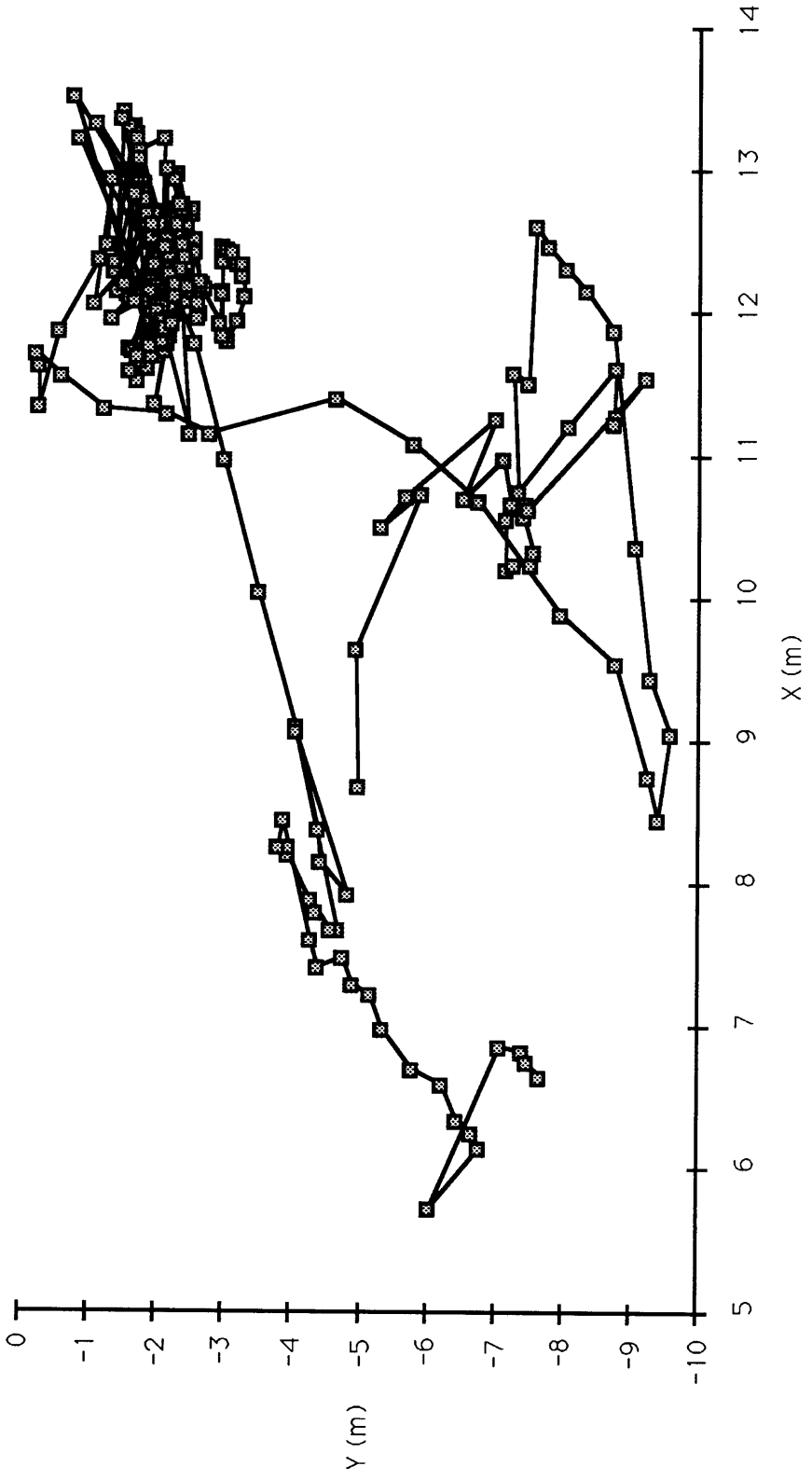


Figure 37. BAT On RMS in Marshall Tank

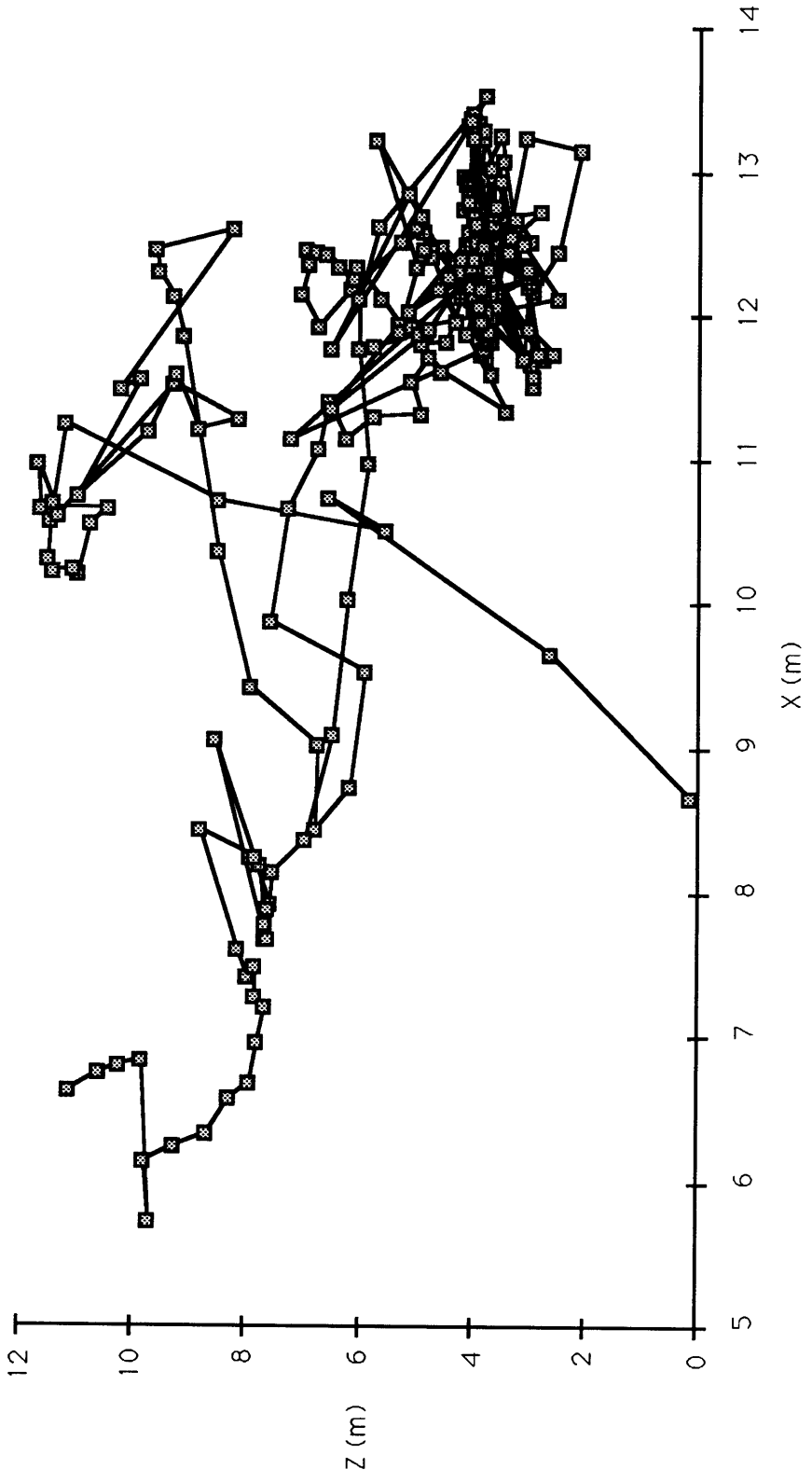


Figure 38. BAT On RMS in Marshall Tank

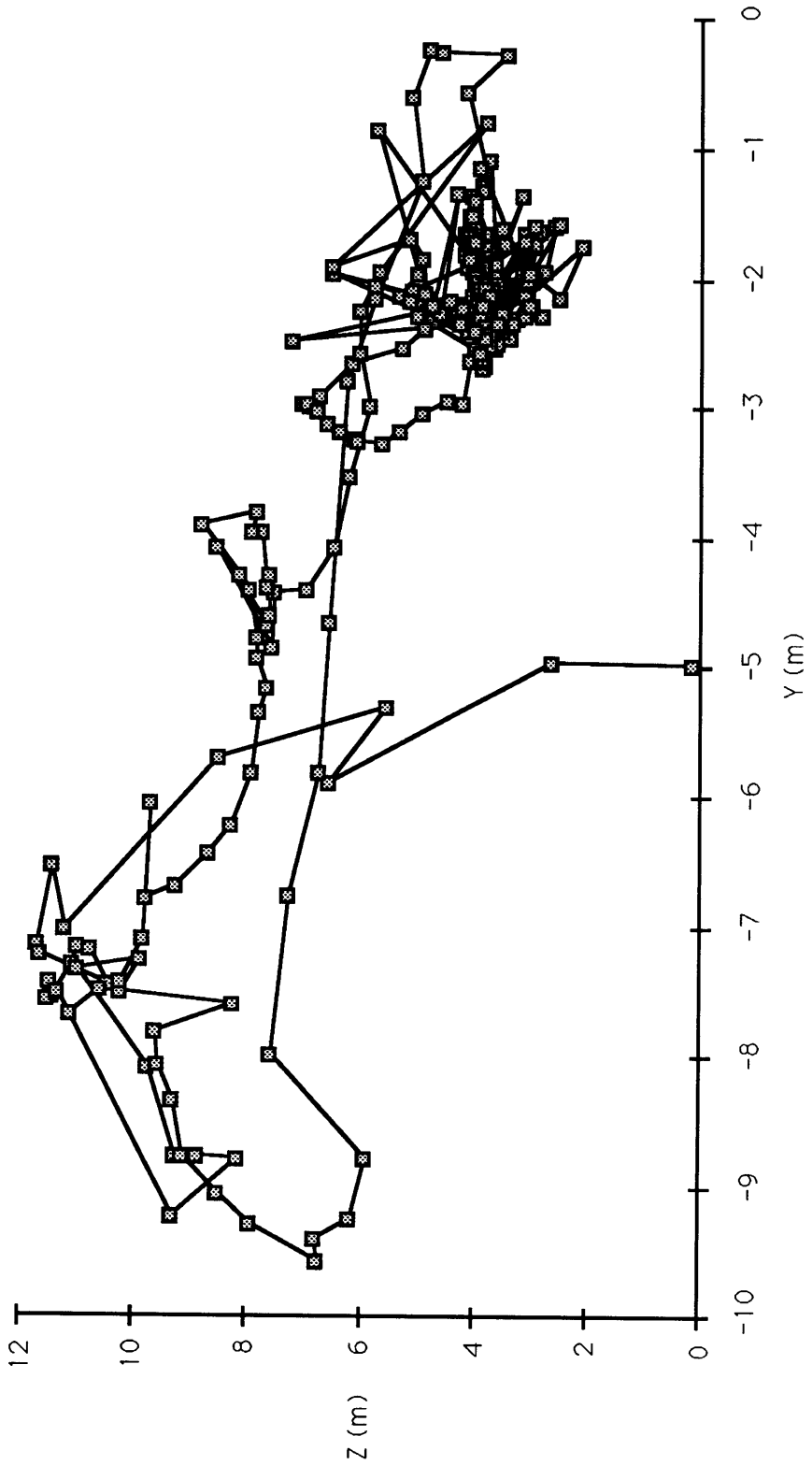


Figure 39. BAT On RMS in Marshall Tank

4.0 Conclusions and Recommendations

The Three-Dimensional Acoustic Positioning System provides position data accurate to within 8 cm (on average) for static situations and for dynamic situations in which the vehicle is moving at less than 1.5 meters per second. For dynamic situations involving higher velocities, the coordinates produced are less accurate. The accuracy of the attitudes provided is directly related to the validity of the coordinates calculated, as errors in range measurements lead to errors in the coordinate calculations, which lead to errors in attitude determinations. A number of applications can directly benefit from the data provided by 3DAPS: any measurement needed within a neutrally-buoyant environment can either be performed or checked with the system, and accurate position and attitude data is extremely useful for controlling telerobotic vehicles.

At present, the system suffers several major disadvantages. Its primary downfalls are accuracy, speed, and robustness. Each of these drawbacks is discussed in the following sections.

4.1 System Accuracy and Speed

The static tests provide the best examples of the behavior of the range-acquisition system used in the receiver. For the most part the raw range data determined by the receiver was within a tolerable margin of error. However, occasional incorrect values were accepted by the system as valid. Since a very large portion of the data acquired was valid and since there is no absolute guarantee that all the data the system acquires will be perfect, the method of range-acquisition used can be tolerated. However, the present algorithm for calculating the three-dimensional positions from the data provided by the receiver is not adequate to deal with significant amounts of erroneous data. Using the method described in section 2.3, if more than one range per hydrophone is incorrect, the coordinates calculated by the software will be in error.

Further, if a significant number of the ranges per hydrophone are zero, it is possible that valid ranges will be ignored in the algorithm's attempt to achieve a minimum c_{hk}. A smaller c_{hk} will be determined if there are fewer ranges necessary to fit to the estimated coordinates.

While the Newton-Raphson method of function minimization is very accurate, its implementation in the coordinate calculation software could be greatly improved. The present method of calculation involves, at the worst, 90 iterations per hydrophone for a possible maximum of 360 iterations per set of range data. The rationale of ignoring one range to achieve a smaller c_{hk} is untenable considering that the algorithm is not utilizing all the data present. In every configuration used in these experiments each hydrophone was in a position that was fixed with respect to the other hydrophones. This information could be used as a set of constraints on the Newton-Raphson calculations, forcing the hydrophones to maintain a specific arrangement with respect to one another. Recent studies (reference 10) have shown that such an algorithm does in fact calculate the coordinates with greater accuracy than the current software, using a set of data from one of the static tests containing a large number of erroneous values. At sample points where the present system calculated incorrect coordinates, the constrained algorithm calculated coordinates that more closely matched the actual measured values. However, as this algorithm was developed

Another major drawback of the current 3DAPS system is the time necessary to acquire range data and calculate coordinates and attitudes. The time needed to acquire the range data at the receiver is dependent upon the fact that all the thumpers are functionally equivalent. The acoustic pulse emitted from one thumper is similar enough to that produced by any other thumper to require thumper sequencing. Decreasing the amount of time between thumper firings is limited by the receiver's capability to acquire valid ranges, which limited the firing delay in these experiments to 0.22 seconds. One possible method of improving the data acquisition rate could be to control the signal emitted by the thumpers. A possible approach to this method

would be to use electrical, rather than mechanical, emitters. The hydrophones used in the current system to receive the acoustic pulse can also be employed to emit acoustic pulses, by sending an electric signal to the hydrophone. This would require a redesign of the sequencer, but such a design could incorporate capabilities not available with the present sequencer, such as microprocessor-based operation. With greater control over the acoustic signal, it may be possible to cause each new thumper to emit a pulse that is different from any other thumper, by using a different frequency for each thumper. All the thumpers could then be fired simultaneously, which would reduce the range-acquisition time by a factor of eight, assuming a full complement of thumpers is used.

A second source of delay is found in the monitoring computer. A six megahertz IBM PC/AT with a math coprocessor cannot compute the coordinates fast enough for a realtime control system. An Apple Macintosh II running at sixteen megahertz and using the algorithm described in section 2.3 does calculate coordinates faster. However, using the constrained algorithm mentioned earlier, the calculations are accomplished faster still, mainly due to the reduction in the number of iterations required to determine accurate coordinates. The use of the constrained Newton-Raphson algorithm is therefore strongly recommended for any further research utilizing data from this system. It is also urgently recommended that whatever monitoring computer is used, it must calculate coordinates in real time. In one of the controlled dynamic tests an anomaly in the range data was noted. During the static positioning phase where the cart was held for static measurements at points along the beam, the range values from four of the thumpers (those along the x-axis) were expected to increase while range values from the remaining four thumpers were expected to decrease. Such behavior was expected because the hydrophones were moving away from the first four thumpers and toward the last four. However, upon reaching the end of the traverse, the ranges associated with one thumper abruptly increased, inconsistent with the actual ranges subsequently measured by divers. The ranges measured by the

system differed from the correct measurements by approximately two meters. More important, however, is that the range measurements made by the system for each of the hydrophones were consistent, with an average standard deviation for ranges to the thumper in question of five millimeters. The reason for this phenomenon was traced to the fact that the receiver uses the coordinates determined by the monitoring computer to set the window of allowable count values, and since the IBM was sending down the same, incorrect, coordinates for each set of ranges, the count window remained the same for every set of ranges. This particular window happened to preclude the valid signal from this thumper as the hydrophones had moved to a position where the valid pulse would be received too early. The valid signal, arriving in time before the counters had reached the minimum window value, was ignored. A later signal, probably a reflection of the valid pulse, arrived within the window and was accepted as valid. Because the receiver had acquired a signal within the window of expected values, the window size was not increased, and so the receiver continued to accept an invalid signal. This set of circumstances can only be eliminated if the monitoring computer is calculating new coordinates with each set of ranges, and thus the use of a Mac II or a significantly faster Intel 80386-based computer is strongly recommended.

4.2 System Robustness

The major obstacles in acquiring data for this thesis were due to the apparent fragility of the component parts of the system. Mechanical and electronic breakdowns and failures in the thumpers, the receiver, and the sequencer caused significant delays and produced unreliable data.

Because of their nature, the thumpers had more failures than any other part of the system. Each thumper contains a moving rod, to which is attached a signal wire, which would become unattached with enough use. Fixing a thumper requires the thumper to be unsealed, opened up, debugged, and resealed. As mentioned in section

4.1, a hydrophone can also be used as an emitter, which would obviate the need for a mechanical acoustic pulse generator, be much more compact, and likely break down less often.

The receiver's major design drawback is that the connection of the hydrophones to their respective range counter cards requires the removal of the other circuit cards in the box. By their nature, the removal and replacement of these cards greatly increases the risk of causing a short or broken circuit where previously none existed. Also, the range counter cards mostly consist of counter chips and combinational logic, all of which could be replaced by small microprocessors or microcontrollers. The fiber optic connections, especially the high-speed line carrying the contact and thumper identification signals from the sequencer, were arranged inside the receiver such that it was possible to damage the ends of the internal fiber optic lines by incautious removal of the decoder/support card. A solution to this problem would be to convert the signal from light to electricity and vice versa right after the signal enters the box, on the lid, and then connect the electrical signal through an electrical connector.

Another problem with the fiber optic system used occured when the high-speed sequencer timing line was not "clean". The receiver chip for this line was not very sensitive, so any attenuation of the signal was unreadable by the chip. A high-speed, high-sensitivity fiber optic line would be more desirable for this part of the system.

If the receiver were redesigned, one recommendation would be to replace its present microprocessor with one that can be programmed and reprogrammed from the monitoring computer while in operation. Such a capability would allow the user to more closely monitor the behavior of the receiver and more easily determine and implement problem solutions from the surface if something goes wrong.

The sequencer in general was more well-behaved than either the thumpers or the receiver. Excluding the catastrophic failure of a single chip, the sequencer was the least troublesome piece of the system. However, some general improvements can still be suggested. When constructed the sequencer was designed and built from

simple combinational logic circuits. By taking advantage of the advances in VLSI, microprocessors, and microcontrollers, the sequencer could be redesigned to be less complex and more robust than its present configuration. Ideally, a microprocessor-based sequencer and receiver would be fully controlled by the monitoring computer, so that the operator of the system need only interact with one device, and be able to command great control over both the sequencer and the receiver.

4.3 Conclusion

Overall, the system at present is good, but could be better. With the preceding suggestions in mind, a faster, more accurate, and more robust system could be designed and implemented for use as position and attitude data input within a feedback control system for neutrally buoyant teleoperated robots. Again, the major limitation on the use of this system as part of a feedback control system is the amount of time needed to acquire the range data. If that can be shortened, the system could become even more useful in neutral buoyancy telerobotics research.

5.0 References

[1] Spofford, J.R., "3-D Position and Attitude Measurement for Underwater Vehicles", SSL Report #21-86, MIT, December 1986.

[2] Kowalski, K.G., "3DAPS: Technical Reference", MIT SSL Report in progress

[3] Brüel and Kjær, "Instruction Manual — Hydrophone Types 8101, 8103, 8104, 8105", October 1986

[4] Onset Computer Corp., "The CPU-8088 Manual", May 1986

[5] Cousins, D., "The Instrumented Beam System", SSL Report #18-86, MIT, October 1986.

[6] Viggh, H.E., "Artificial Intelligence Applications in Teleoperated Robotic Assembly of the EASE Space Structure", S.M. Thesis, Dept. of Aeronautics and Astronautics, MIT, February 1988.

[7] Vyhnalek, G.G., "A Digital Control System for an Underwater Space Simulation Vehicle Using Rate Gyro Feedback", S.M. Thesis, Dept. of Aeronautics and Astronautics, MIT, June 1985.

[8] Rowley, V.R., "Effects of Stereovision and Graphics Overlay on a Teleoperator Docking Task", S.M. Thesis, Dept. of Aeronautics and Astronautics, MIT, September 1989

[9] Atkins, E.M., "Autonomous Flight and Docking of an Underwater Robotic Vehicle During Neutral Buoyancy Simulation of Satellite Servicing", S.M. Thesis in progress, Dept. of Aeronautics and Astronautics, MIT.

[10] Sanner, R.M., "Optimum Position and Attitude Computation Using 3DAPS Range Data", MIT SSL Report in progress.

[11] Spofford, J.R., "Beam Assembly Teleoperator: Coordinates and Kinematics", SSL Report #6-87, MIT, April 1987.

6.0 Appendices

6.1 Appendix A. MIT Pool and Marshall Tank Thumper Positions

Tables A-1 and A-2 list the x, y, and z coordinates of the thumpers as deployed in both the MIT pool and the Marshall tank, in their respective coordinate systems as defined in section 2.1.

Thumper	X	Y	Z	Error
1	1006	255	150	10
2	998	255	3093	10
3	10871	255	168	10
4	10854	255	3098	10
5	1003	12605	166	10
6	1007	12605	3090	10
7	10872	12605	166	10
8	10878	12605	3088	10

All measurements in mm.

Table A-1. Pool Test Thumper Locations

Thumper	X	Y	Z	Error
1	1490	-8485	6850	10
2	12367	-10671	11030	10
3	18608	-5013	4735	10
4	17980	6053	11050	10
5	-2427	-1018	11040	10
6	304	7113	4760	10
7	5033	10439	8920	10
8	10760	10732	4790	10

All measurements in mm.

Table A-2. Marshall Tank Thumper Locations

6.2 Appendix B. 3DAPS Operating Software

The three sets of sourcecode listed here are the programs used to run and operate the 3DAPS system.

The first set is the assmebly language program that operates the receiver. It consists of five sets of code: the receiver main routines (RX1A), two included subroutine packages (RX_INC.CS1, RX_INC.CS2), the 8088 data segment initializations (RX_INC.DS), and a routine for performing square-roots (RX_INC.SQR).

The second set of code is the program run on the IBM while 3DAPS experiments were being conducted (M3DAPSA.C). The next set is the program used to calculate in non-realtime the coordinates and attitudes (M3FREADE.C) from the range data sets as produced by M3DAPSA.C, and the last set is the code developed to simulate the controlled dynamic experiment (SIMULATR.C).

RX1.ASM:

```
title 03/08/87 11:33 ******* RX1A  JRS  3DAPS   ********
; modified by Matt Machlis for multi-pt receiver, Summer 1988
; last modification 7/20/88
page 59,96
RX      equ     '8'
;
onboard equ     0F000h          ; onboard segment address
pio     equ     0000h           ; parallel I/O offset & RAM
sio     equ     4000h           ; serial I/O offset
rtc     equ     6000h           ; real time clock offset
wait    equ     8000h           ; wait mode offset
hyb     equ     0A000h          ; hybernate mode offset
rom     equ     0C000h          ; eprom offset
reset   equ     0FFF0h          ; reset vector
CR      equ     0Dh             ; carriage return
LF      equ     0Ah             ; linefeed
;
X               equ     0
Y               equ     2
Z               equ     4
;
;=================================================
; ---- Page zero RAM allocation ----
;=================================================
rx1a_ds segment         at 0000h
                org     0000h
;       Interrupt vectors look like seg=0 to CPU
divzero_i       dd      ?               ;0 divide by zero vector
sstep_i         dd      ?               ;1 single step vector
nmir_i          dd      ?               ;2 NMI vector
onebyt_i        dd      ?               ;3 one byte vector
ovrflo_i        dd      ?               ;4 overflow vector
;
numer           dw      ?       ;cnt->mm fraction
denom           dw      ?       ; "
nt              db      ?       ;number of thumpers (1-8)
verfl           db      ?       ;flag for verification of setup data read
r_cm            dw      ?       ;range in cm
r_x             dw      4 dup(?)  ;x coordinate, signed
r_y             dw      4 dup(?)  ;y coordinate, signed
r_z             dw      4 dup(?)  ;z coordinate, signed
range           dw      8*4 dup(?) ;measured thumper ranges
est_r           dw      8*4 dup(?)  ;estimated ranges
t_coord         dw      8*3 dup(?) ;[NT][3] thumper coord array
temp4           dd      ?       ;temp var
gate_width      dw      ?       ;minimum half-width of gates (in
count)
g_widths        dw      8*4 dup(?) ;dynamic half-width values (in
count)
div_sign        db      ?       ;+ or - division flag for recovery
ustat           db      ?       ;uart status flags
max_rc          db      8*4 dup(?) ;max range cnt
min_rc          db      8*4 dup(?) ;and the min
spaz            db      ?       ;action counter
cond            db      4 dup(?)  ;range ctr status
thump           db      ?       ;thumper id
rc_lo           db      ?       ;used as counter to index lo rc gates
rc_hi           db      ?       ;used as counter to index hi rc gates
rc_stat         db      ?       ;used as counter to index rc stat bytes
_dec_v          db      6 dup(?)  ;6 digit ascii workspace (inc +/-)
                org     1FFFh   ;top of offboard ram
mstack          equ     $               ; top of monitor stack
rx1a_ds ends
;-----------------------
INCLUDE                 RX_INC.DS       ;ONSET board memory
definitions
;-----------------------
rx1a_cs segment         byte public 'CODE'
assume  cs:rx1a_cs, ds:rx1a_ds, es:rx_inc;;;, ss:rx1a_ss
        org     0       ;at start of segment!
;
;=================================================
; --- PROGRAM ENTRY AND INITIALIZATION ----
;-------------------------------------------------------
init:   mov     ax,0            ; initialize segment regs
        mov     ds,ax           ;
        mov     ss,ax
        mov     ax,onboard      ;
        mov     es,ax           ;
        mov     sp,offset mstack ; pio ram
        ;----- see if this is a warm restart or a cold boot
        cmp     divzero_i, offset d0_handler            ;interrupt vector
        jne     cold_b
        mov     ax, cs
        cmp     divzero_i[2], ax                ;code seg
        jne     cold_b
        ;----- warm restart
;       call    reenter_3d      ;go to the predefined startup location
cold_b:
;-----------------------
```

```
INCLUDE                 RX_INC.CS1      ;ONSET startup code
;-----------------------
signon: mov     divzero_i, offset d0_handler        ;interrupt vector
        mov     ax, cs
        mov     divzero_i[2], ax            ;code seg
        jmp     monitr  ;
;
;=======================================================
monitr  label   near    ; MONITOR MAIN ENTRY
;-------------------------------------------------------
        mov     ax,0            ; initialize segment regs
        mov     ss,ax           ; this all allows warm restarts
        mov     ds,ax           ;
        mov     ax,onboard
        mov     es, ax          ;offboard on multipoint, onboard 1-pt
        mov     sp,offset mstack ; and reset stack
        mov     al, '+'
        call    sndbyt          ;send running ok signal topside
        jmp     main
;
;=======================================================
;------ 3DAPS constants and macros
RC1_STAT        EQU     000100b         ;
RC1_LO                  EQU     000101b         ;
RC1_HI          EQU     000110b         ;
RC4_STAT        EQU     010000b         ;
RC4_LO                  EQU     010001b         ;
RC4_HI          EQU     010010b         ;
RC_ADD                  EQU     000100b         ;
DECODE                  EQU     000000b         ;
LATSTB                  EQU     100000b         ;to strobe a
receiver latch
erofset         equ     112
;
latch           macro   rcaddr, outdat, indat
                mov     [piopc], rcaddr             ;set up latch addr
                mov     [piopb], outdat             ;data on output bus
(portB)
                xor     [piopc], LATSTB  ;strobe pc5 bit high
                mov     indat, [piopa]              ;read input bus
(portA)
                xor     [piopc], LATSTB  ;unstrobe pc5 bit
                endm
;
checkchar       macro           ;sets ZERO if char is available
                mov     al,siocs  ; get current status
                and     al,udr+utbre        ; mask all but RX, TX ready
                or      al,ustat  ; or in old status
                mov     ustat,al  ; and save
                and     al,udr              ; is byte available
                endm
;
idiv_0          macro   arg     ;set sign flag for overflow recovery
                local   pos, nxt, doit
                cmp     dx, 0               ;check dividend
                jge     pos
                mov     div_sign, +1        ;set neg
                jmp     nxt
        pos:    mov     div_sign, 1         ;set pos
        nxt:    cmp     arg, 0              ;check divisor
                jge     doit
                neg     div_sign            ;toggle sign
        doit:   idiv    arg
                endm
;
limit           macro   arg     ;limit result of 16bit add/sub
                local   beneg, isok
                jno     isok                                    ;check
for overflow
                jns     beneg                                   ;
                mov     arg, 7FF0h                              ;max pos
                jmp     isok                                    ;
        beneg:  mov     arg, -7FF0h                             ;max neg
        isok:
                endm
;
cnt2mm          macro           ;converts AX from count to mm
                mul     numer
                div     denom
                endm
;
mm2cnt          macro           ;converts AX from mm to count
                mul     denom
                div     numer
                endm
;
teletype macro  stroff
;               mov     dx, offset stroff
;               call    sndstg
                endm
;
;=======================================================
```

```
;Main program begins here
;-------------------------------------------------------
main:           call    getkey          ;get code letter
                cmp     al, 'A'         ;check if setup data is being
sent
                jnz     mn1
                call    setup
                jmp     main
mn1:            cmp     al, 'P'         ;check if starting thumper
cycle
                jnz     main
                call    doacyc
                jmp     main
;
;=======================================================
setup   proc    near    ;read & verify setup data from topside PC
;-------------------------------------------------------
begset:         call    getword
                mov     gate_width, bx  ;half-width of RC count
gates
                call    getword
                mov     numer, bx       ;numerator for cnt2mm
conversion
                call    getword
                mov     denom, bx       ;denominator for cnt2mm
conversion
                call    getbyt
                mov     nt, al          ;number of thumpers (1-8)
                mov     dx, 0           ;index to thumper #
thmploop:       mov     si, dx
                add     si, dx
                add     si, dx
                add     si, si          ;set si=dx*6
                call    getword
                mov     t_coord[si], bx ;set X coord of thumper #dx
                inc     si
                inc     si
                call    getword
                mov     t_coord[si], bx ;set Y coord of thumper #dx
                inc     si
                inc     si
                call    getword
                mov     t_coord[si], bx ;set Z coord of thumper #dx
                inc     dl
                cmp     dl, 8           ;check if all thumpers read
in
                jnz     thmploop
;begin second read cycle for verification
                mov     verfl, '+'      ;set verify flag to plus
                call    getword
                cmp     bx, gate_width  ;verify half-width of RC
count gate
                jz      ver0
                mov     verfl, '-'
ver0:           call    getword
                cmp     bx, numer       ;verify cnt2mm numerator
value
                jz      ver1
                mov     verfl, '-'
ver1:           call    getword
                cmp     bx, denom       ;verify cnt2mm
denominator value
                jz      ver2
                mov     verfl, '-'
ver2:           call    getbyt
                cmp     al, nt          ;verify # thumpers value
                jz      ver3
                mov     verfl, '-'
ver3:           mov     dx, 0           ;index to thumper #
thmploop2:      mov     si, dx
                add     si, dx
                add     si, dx
                add     si, si          ;set si=dx*6
                call    getword
                cmp     t_coord[si], bx ;check X coord of thumper
#dx
                jz      thlp1
                mov     verfl, '-'
thlp1:          inc     si
                inc     si
                call    getword
                cmp     t_coord[si], bx ;check Y coord of thumper
#dx
                jz      thlp2
                mov     verfl, '-'
thlp2:          inc     si
                inc     si
                call    getword
                cmp     t_coord[si], bx ;check Z coord of thumper
#dx
                jz      thlp3
                mov     verfl, '-'
```

```
thlp3:          inc     dl
                cmp     dl, 8           ;check if all thumpers
verified
                jnz     thmploop2
                mov     al, verfl
                call    sndbyt
                cmp     al, '-'
                jnz     thskip
                jmp     begset
thskip:         mov     ax, gate_width
                mov     si,0            ;init g_widths loop index
gwloop:                 mov     g_widths[si],ax    ;initialize g_w[si]
                inc     si
                cmp     si,32           ;see if done with loop
                jne     gwloop          ;if not, continue
                ret
setup           endp
;
;=============================================================
doacyc  proc    near    ;go through one cycle of everything
;-------------------------------------------------------
;
                ;read positions for 4 hydrophones from topside PC
posstart:       mov     dx, 0           ;index to hydrophone #
                mov     cl, 0           ;reset checksum register
posloop:        mov     si, dx
                add     si, si          ;index*2
                call    getword         ;get x coord for this
hydrophone
                mov     r_x[si], bx
                add     cl, al
                call    getword         ;get y coord for this
hydrophone
                mov     r_y[si], bx
                add     cl, al
                call    getword         ;get z coord for this
hydrophone
                mov     r_z[si], bx
                add     cl, al
                inc     dl
                cmp     dl, 4
                jnz     posloop                 ;run loop for next
hydrophone
                call    getbyt          ;get checksum
                cmp     al, cl          ;see if they match
                jz      posok
                mov     al, '-'         ;no match
                call    sndbyt          ;send minus
                jmp     posstart ;and get positions again
posok:          mov     al, '+'
                call    sndbyt          ;send plus
;
                ;compute ranges from each thumper to each hydrophone
                mov     di, 0           ;index to hydrophones
hyd_loop:       mov     si, 0           ;3 word thumper index
                mov     cx, 0           ;1 word thumper index
cr_loop:        mov     ax, t_coord[si+X] ;get Xi
                mov     bx, di
                add     bx, bx          ;bx=2*di
                sub     ax, r_x[bx]
                imul    ax              ;square
                mov     temp4, ax
                mov     temp4[2], dx    ;done with X
                mov     ax, t_coord[si+Y] ;get Yi
                mov     bx, di
                add     bx, bx          ;bx=2*di
                sub     ax, r_y[bx]
                imul    ax              ;square
                add     temp4, ax
                adc     temp4[2], dx    ;done with Y
                mov     ax, t_coord[si+Z] ;get Zi
                mov     bx, di
                add     bx, bx          ;bx=2*di
                sub     ax, r_z[bx]
                imul    ax              ;square
                add     ax, temp4       ;result in DX:AX
                adc     dx, temp4[2]    ;done with Z
                ;--- root & save
                and     dx, 3fffh       ;force 30 bit argument
                call    sqrt_30
                mov     bx, cx
                add     bx, bx
                add     bx, bx          ;set bx=cx*4
                add     bx, di          ;and add hydrophone index
                add     bx, bx          ;and double the result
                mov     est_r[bx], ax   ;save calculated range
                ;--- check end of loop
                mov     bl, nt
                dec     bl
                cmp     cl, bl
                jnz     cr_more         ;do inner loop for next
thumper
```

```
                inc     di
                cmp     di, 4
                jnz     hyd_loop        ;do outer loop for next
hydrophone
                jmp     dethilo
cr_more:        mov     ax, 6
                add     si, ax          ;3 word inc
                inc     cx              ;1 word inc
                jmp     cr_loop
;
                ;determine high and low range gate values for all 32
cases
dethilo: mov    thump, 0        ;index to thumpers
thploop:        mov     dx, 0           ;index to hydrophones
thploop2:       mov     bl, thump
                mov     bh, 0
                mov     si, bx
                add     si, si
                add     si, si          ;si=thump*4
                add     si, dx          ;add hydrophone #
                mov     di, si          ;save to di
                add     si, si          ;double si
                mov     ax, est_r[si]   ;get estimated range
                push    dx
                add     ax, erofset     ;correct for const range error
                mm2cnt                  ;convert back to count
                pop     dx
                mov     cx, ax          ;hide estimated count
                add     ax, g_widths[si] ;compute max count
                add     ax, 10000000b   ;add one to 8th highest bit
                jns     noclpl          ;negative or overflow?
                mov     ax, 7f00h       ;highest allowable max
count
noclpl:         add     ax, ax          ;shift for upper 15 bits
                mov     max_rc[di], ah  ;store max range (top 8)
                mov     ax, cx          ;get estimated count again
                sub     ax, g_widths[si] ;compute min range
                js      fixlow          ;negative or overflow?
noclp2:         cmp     ax, 0100h       ;check for gate below 02
                jae     low_ok
fixlow:         mov     ax, 0100h       ;lowest allowable min count
low_ok:                 add     ax, ax          ;shift for upper 15
bits
                mov     min_rc[di], ah  ;store min range (top 8)
                inc     dx              ;increment hydrophone
index
                cmp     dx, 4
                jnz     thploop2        ;do outer loop for next RC
                inc     thump
                mov     al, thump
                cmp     al, nt
                jnz     thploop
;
                ;zero out range matrix
                mov     si, 0
zroloop:        mov     range[si], 0
                inc     si
                inc     si
                cmp     si, 64
                jnz     zroloop
;
                ;do a cycle!
                call    cycle
;
                ;go through and update gate-widths matrix based on
                ;results of cycle
                mov     si,0            ;initialize loop index
gwbegl:         mov     ax, g_widths[si] ;get a gate-width
                cmp     range[si],0     ;see if corresponding
range=0
                jne     gwskip1                 ;if not, don't
double
                add     ax, ax          ;double gate-width
                jmp     gwnxt           ;continue loop
gwskip1:        cmp     ax, gate_width  ;see if at min. width value
                je      gwnxt           ;if so, continue loop
                shr     ax, 1           ;halve ax
gwnxt:          mov     g_widths[si], ax ;save gate-width
                inc     si              ;increment loop index
                cmp     si,32           ;check if done
                jne gwbegl              ;nope-continue loop
;
                ;send results topside
sndlbeg:        mov     dl, 0           ;checksum storage
                mov     si, 0           ;index to range[]
sndloop:        mov     bx, range[si]
                add     dl, bl
                add     dl, bh          ;update checksum
                mov     al, bh
                call    sndbyt          ;send high byte of current
range
                mov     al, bl
```

```
                call    sndbyt          ;send low byte of current
range
                inc     si
                inc     si
                mov     al, nt
                mov     ah, 0
                mov     di, ax
                add     di, di
                add     di, di
                add     di, di          ;di=nt*8
                cmp     si, di          ;see if have sent all ranges
                jnz     sndloop         ;continue sending ranges
                mov     al, dl
                call    sndbyt          ;send checksum
                call    getbyt          ;get flag
                cmp     al, '+'         ;check if okay
                jz      snddone
                call    delay2
                jmp     sndlbeg         ;no good - try sending again
;
snddone:        ret
doacyc          endp
;
;
;================================================
cycle   proc    near    ;READ ALL RANGES AND TYPE 'EM
;-------------------------------------------------------
                mov     spaz, 0         ;wait for ch0 flag
                ;--- wait for counter to start
cycloop:        latch   RC1_STAT, 0, al ;get status into AL
                and     al, 111b        ;isolate "run/pastlo" bits
                xor     al, 111b
                jz      cycloop         ;keep waiting if still
stopped
                ;--- get thumper no.
                latch   DECODE, al, al
t_thpr_o:       and     al, 7           ;mask
                cmp     spaz, 0         ;waiting for ch0?
                jne     cyc_on          ;if not
                cmp     al, 0           ;yes, is it ch0?
                jne     cycloop         ;try again if not
                mov     spaz, 0FFh      ;reset flag and fall thru to
cyc_on
cyc_on:         mov     thump, al
                ;--- set range gate for this channel
                mov     ah, 0
                mov     si,ax           ;si is index into min_rc &
max_rc
                add     si,si
                add     si,si           ;here si equals thump*4
                mov     rc_lo,RC1_LO    ;initialize gate value indices
                mov     rc_hi,RC1_HI
mxmnloop:       mov     al, min_rc[si]
                mov     bl, rc_lo
                latch   bl, al, al      ;set low gate value
                mov     al, max_rc[si]
                mov     bl, rc_hi
                latch   bl, al, al      ;set high gate value
                cmp     rc_lo, RC4_LO   ;see if set ranges set on all 4
RCs
                jz      mxmnend                 ;if so, exit loop
                add     rc_lo, RC_ADD   ;update gate value indices
                add     rc_hi, RC_ADD
                inc     si              ;update min_rc/max_rc
index
                jmp     mxmnloop
                ;--- wait for counters to finish
mxmnend:        mov     rc_stat, RC1_STAT
                mov     si,0                    ;index to COND[]
cyc02:          mov     bl, rc_stat
                latch   bl, 0, al       ;get status into AL
                and     al, 00000111b           ;filter out RUN*
bit
                xor     al, 111b
                jnz     cyc02                   ;keep waiting if
not done
                latch   bl, 0, al
                mov     cond[si], al            ;save status
                inc     si
                cmp     si,4                    ;check if last RC
done
                jz      cycl05
                add     rc_stat, RC_ADD         ;update rc_stat to
next RC
                jmp     cyc02                   ;execute loop for
next RC
                ;--- get count value, compute range
cycl05:         mov     si,0                    ;si=RC card index
counter
                mov     rc_lo, RC1_LO
                mov     rc_hi, RC1_HI
cycl01:         mov     al, thump
```

```
                mov     ah, 0
                mov     di, ax
                add     di, di
                add     di, di
                add     di, si              ;di=thump*4+si
                mov     al, min_rc[di]      ;set range latches,
                mov     cl, rc_lo
                latch   cl, al, bl          ;and get count
                mov     al, max_rc[di]
                mov     cl, rc_hi
                latch   cl, al, bh
                and     bx, 7FFFh           ;15-bit count value
t_rnge_o:       mov     ax, bx              ;convert to mm
                cnt2mm
                mov     r_cm, ax            ;save in temp variable
                ;--- check status, type thpr# & range, display & save
                mov     al, cond[si]                ;check worth of
count
                test    al, 00100000b               ;is it good?
                jz      itisokc
                test    al, 00010000b               ;is it late?
                jz      itsltc
itsovrc:  mov   r_cm, 0             ;assume overrun
                jmp     itisokc
itsltc:         mov     r_cm, 0             ;count was late -- ignore it
itisokc:  mov   ax, r_cm            ;get temp back
                add     di, di
                mov     range[di], ax       ;save for coords
                cmp     si,3                ;see if have saved all 4
ranges
                jz      itsdunc             ;for this thumper
                inc     si
                add     rc_lo, RC_ADD
                add     rc_hi, RC_ADD
                jmp     cycl01
                ;--- check for last channel, else loop
itsdunc:  mov   al, thump
                inc     al
                cmp     al, nt
                je      done_cyc
                jmp     cycloop
done_cyc:       ret                 ;done with one thumper cycle
cycle           endp
;
;
;================================================
```

```
d0_handler      proc    far         ;DIVIDE-BY-ZERO INTERRUPT
HANDLER
;-------------------------------------------------------
                cmp     div_sign, 0
                jg      pos_fix
                je      say_what
                mov     ax, -7FF0h          ;neg max
                mov     dx, 0
                iret
pos_fix:        mov     ax, 7FF0h           ;pos max
                mov     dx, 0
say_what:       iret
d0_handler      endp
;
;=======================================================
time_delay      proc    near        ;DELAY AL SECONDS
;-------------------------------------------------------
        push    bx
        mov     rtccsr, 0001b       ;clock running
t_loop: mov     bl, seconds         ;get ref sec
t_wait: mov     bh, seconds         ;get now sec
        cmp     bl, bh
        je      t_wait              ;wait for new sec
        dec     al
        jnz     t_loop              ;keep waiting
        pop     bx
        ret
time_delay      endp
;
;
;-----------------------
INCLUDE                 RX_INC.SQR      ;Square root procedure
SQRT_30
;-----------------------
;-----------------------
INCLUDE                 RX_INC.CS2      ;Console & display utilities
;-----------------------
s_cmping        db      'Computing estimated distances...',CR,LF,'$'
s_gating        db      'Computing gating hi/lo values...',CR,LF,'$'
s_zroing        db      'Zeroing out range matrix...',CR,LF,'$'
s_ccling        db      'Cycling through thumpers...',CR,LF,'$'
s_wting         db      'Waiting for CONTACT...',CR,LF,'$'
s_cnt           db      'CONTACT recieved...',CR,LF,'$'
;
s_thump                 db      'thumper=$'
s_hyd           db      ' hydrophone=$'
```

```
;=============================================
;
rx1a_cs  ends
;
         end
```

RX_INC.CS1:

```
;RX_INC.CS1
;INCLUDE FILE FOR 3DAPS RECEIVER CODE (RX1A,RX3)
;at start of code segment
;
; ---- INIT 82C55 PIO ----
; Set port A for input (has pullups), ports B and C for out.
; note: hardware reset puts all ports in input mode.
;
         mov     al,piopb          ; first read ports in case
         mov     ah,piopc          ;  used as inputs
         mov     piocsr,painp+pboutp+pcoutp+tstop
         mov     piopb,al          ; now set outputs to not
         mov     piopc,ah          ;  conflict with lines
;
; ---- INIT 58274 REAL-TIME CLOCK ----
;
         mov     rtccsr,0001b      ; norm,clk on, rupt,no rupts
         mov     rtcmode,0001b     ; 24 hour mode,00 leap year
;
; ---- INIT 8252 SIO & SET BAUD RATE ON INCOMMING <cr> ----
         mov     siocs,eightb+nopar+onestop
         mov     siobrs,b9600
         mov     siomcr,mdmdi+rcvren+siodi+dtrhigh
         mov     ustat,utbre
;
_surt0:  call    getbyt            ; get something from UART
         cmp     al,CR             ; if match
         jz      signon            ;  then already set
         mov     ah,b4800          ; assume 4800
         cmp     al,0e6h           ;  get constant
         jz      _surt1            ;
         mov     ah,b2400          ; assume 2400
         cmp     al,078h           ;
         jz      _surt1            ;
         mov     ah,b1200          ; assume 1200
         cmp     al,080h           ;
         jz      _surt1            ;
         mov     ah,b300                   ; assume 300
         cmp     al,00h            ;
         jz      _surt1            ; try again if no match
         mov     ah,b19200         ; assume 19.2K
         cmp     al,0f0h           ; test result
         jl      _surt0            ; must be 19.2 or greater
;
```

```
_surt1:  mov     siobrs,ah       ; insert baud constant
         call    delay           ; delay a bit
         mov     al,siodat       ; get char to clear
         mov     ustat,utbre     ; fall through to signon
```

RX_INC.CS2:

```
;RX_INC.CS2
;INCLUDE FILE FOR 3DAPS RECEIVER CODE (RX1A,RX3)
;somewhere in code segment
;=================================================
sndstg   proc    near     ; SEND STRING (&DX) OUT UART
; Send the '$' terminated string pointed to by CS:DX to the
; UART.  Delay after sending line-feeds for slow terminals.
; ALTERS: AL=?
;-------------------------------------------------
         push    di              ; save reg
         mov     di,dx           ; transfer pointer
_sstg0:  mov     al,CS:[di]      ; next char to al rel to CS!
         inc     di              ; ready next position
         cmp     al,'$'          ; dollar terminator
         jz      _sstg2          ; yes, exit
         call    sndbyt          ; else display char
         jmp     _sstg0          ; and loop
;
_sstg2:  mov     dx,di           ; adjust pointer
         pop     di              ; restore
         ret                     ; and return
sndstg   endp
;
;=================================================
snddec   proc    near     ;TYPE DECIMAL TO SERIAL
;alters: AL
;-------------------------------------------------
         mov     al, [_dec_v+5]          ;get sign
         call    sndbyt
         mov     al, [_dec_v+4]          ;get top dec digit
         call    sndbyt
         mov     al, [_dec_v+3]          ;all 5 digits...
         call    sndbyt
         mov     al, [_dec_v+2]
         call    sndbyt
         mov     al, [_dec_v+1]
         call    sndbyt
         mov     al, [_dec_v]
         call    sndbyt
         ret
snddec   endp
;
;
;=================================================
```

```
echocmd         proc    near    ; ECHO CHARACTER IN AL, PRINT
CR-LF
; Echo the command character in AL to the UART, and start a
; new line.
; ALTERS: AL=?
;---------------------------------------------------------
        call    sndbyt          ; echo character
        call    sndcr           ; and start new line
        ret
echocmd         endp
;
;
;=========================================================
sndhxn  proc    near    ; SEND ONE CHARACTER NIBBLE IN AL
; Send the value in the lower four bits of AL to the UART
; as an ASCII hex digit.
; ALTERS: AL=?
;---------------------------------------------------------
        and     al,0fh          ; mask high nibble
        add     al,'0'          ; convert binary to decimal
        cmp     al,'9'          ; is it decimal digit
        jle     _shxn0          ; yes, send and exit
        add     al,7            ; else convert to ASCII hex
_shxn0:         call    sndbyt          ; send and exit
        ret
sndhxn  endp
;
;
;=========================================================
sndhxb  proc    near    ; SEND TWO CHARACTER HEX BYTE IN AL
; Send the value in AL to the UART as two hex digits.
; ALTERS: nothing
;---------------------------------------------------------
        push    ax              ; save value
        rcr     al,1            ; get ms nibble
        rcr     al,1            ;  into ls position
        rcr     al,1            ;
        rcr     al,1            ;
        call    sndhxn          ; send ms nibble
        pop     ax              ; restore original
        call    sndhxn          ; send ls and exit
        ret
sndhxb  endp
;
;
;=========================================================
```

```
sndhxw  proc    near    ; SEND FOUR CHAR HEX WORD IN BX
; Send the value in BX to the UART as four hex digits.
; ALTERS: AL=?
;---------------------------------------------------------
        mov     al,bh           ; get high byte
        call    sndhxb          ; display it
        mov     al,bl           ; get low byte
        call    sndhxb          ; display it and exit
        ret
sndhxw  endp
;
;
;=========================================================
sndcr   proc    near    ; PRINT CARRIAGE RETURN / LINE FEED
; Send a carriage-return line-feed sequence to the UART and
; NO delay for slow terminals.
; ALTERS: AL=?
;---------------------------------------------------------
        mov     al,0dh          ; send carriage return
        call    sndbyt          ;
        mov     al,0ah          ; send line feed
        call    sndbyt          ;
        ;;call  delay           ; NO delay 50 ms for line feed
        ret                     ; and return
sndcr   endp
;
;
;=========================================================
sndspc  proc    near    ; PRINT SPACE CHARACTER
; Send one space character (ASCII 32) to the UART.
; ALTERS: AL=?
;---------------------------------------------------------
        mov     al,' '          ; space over
        call    sndbyt          ; send char
        ret                     ; and return
sndspc  endp
;
;
;=========================================================
sndbyt  proc    near    ; SEND BYTE IN AL OUT UART
; Send the byte in AL out the UART.
; ALTERS: nothing
;---------------------------------------------------------
_sbyt0: push    ax              ; save char on entry
_sbyt1: mov     al,siocs ; get current flags
        and     al,udr+utbre    ; only care RX and TX ready
```

```
        or      al,ustat  ; or in old status
        mov     ustat,al  ; and save
        and     al,utbre  ; is transmit ready ???
        jz      _sbyt1            ; no, loop till
        and     ustat,udr         ; clear xmit ready (wont be)
        pop     ax                ; restore character to send
        mov     siodat,al         ; send char in al
        ret                       ; and return
sndbyt  endp
;
;
;=============================================================
gethxw  proc    near      ; INPUT AND QUALIFY HEX WORD TO BX
; Input hexadecimal digit and form an unsigned word value
; from the last four digits into BX. Terminate input on
; receipt of a carrige-return or colon. Null entry returns
; zero in BX. On return AL contains either FFH for <cr>
; terminated entry, or a colon.
; ALTERS: AL=FFH or ':', BX=word value
;-------------------------------------------------------
_ghxw0:         mov     bx,0              ; clear offset accum
_ghxw1:         call    gethxn            ; get hex nibble
        cmp     al,0ffh           ; was it <cr> ???
        jnz     _ghxw2            ; no, continue
        ret                       ; else return FF
_ghxw2:         cmp     al,':'            ; was it segment/offset sep.
        jnz     _ghxw3            ; no, continue
        ret
;
_ghxw3:         add     bx,bx             ; shift total to make
        add     bx,bx             ;  room for new nibble
        add     bx,bx             ;
        add     bl,al             ; or in new nibble
        call    sndhxn            ; echo keystroke
        jmp     _ghxw1            ; and loop for more
gethxw  endp
;
;
;=============================================================
gethxb  proc    near      ; INPUT AND QUALIFY HEX BYTE TO AL
; Input the next two hex characters and form into byte
; value in AL with no echo. (used by hex loader routine)
; ALTERS: AL=value
;-------------------------------------------------------
        push    cx                ; save reg
        call    gethxn            ; get first nibble
        mov     cl,al             ; clear accum
        add     cl,cl             ; shift into msb position
        add     cl,cl             ;
        add     cl,cl             ;
        add     cl,cl             ;
        call    gethxn            ; get lsb
        add     al,cl             ; and add it in
        pop     cx                ; restore
        ret                       ; and return
gethxb  endp
;
;
;=============================================================
gethxn  proc    near      ; INPUT AND QUALIFY HEX DIGIT TO AL
; Input the next hex character and form a four bit nibble
; in AL. Returns either a valid hex nibble value (0-15) or
; FFH if carriage return or ':' if colon received.
; ALTERS: AL=value or FFH or ':'
;-------------------------------------------------------
_ghxn0:         call    getkey            ; get upper case char
_ghxn1:         mov     ah,al             ; save in ah
        cmp     al,0dh            ; <cr> terminator
        jnz     _ghxn3            ; no, continue
        mov     al,0ffh           ; translate <cr> to FF
        ret                       ; and return it
;
_ghxn3:         cmp     al,':'            ; segment/offset separator
        jnz     _ghxn4            ; no, continue
        ret                       ; else return it
_ghxn4:         sub     al,'0'            ; convert char to binary
        jc      _ghxn0            ; was invalid, try again
        cmp     al,10             ; decimal digit ???
        jc      _ghxn5            ; yes, continue
        sub     al,7              ; fix hex conversion
        jc      _ghxn0            ; invalid, try again
        cmp     al,16             ; test upper limit
        jnc     _ghxn0            ; invalid, try again
_ghxn5:         ret                       ; and return
gethxn  endp
;
;
;=============================================================
getkey  proc    near      ; GET KEY, UPPER CASE TRANSLATE
; Input a byte from the UART to AL, convert lower case to
; upper, pass all punctuation.
```

```
; ALTERS: AL=character
;-------------------------------------------------------
        call    getbyt              ; get a byte
_gkey0:         and     al,7fh              ; force ASCII
        cmp     al,'a'              ; upper case or punct ???
        jc      _gkey1              ; yes, return it
        cmp     al,'z'+1    ; high punctuation ???
        jnc     _gkey1              ; yes, return it
        sub     al,'a'-'A'  ; else convert to upper case
_gkey1:         ret                             ; and return
getkey  endp
;
;
;=======================================================
getbyt  proc    near    ;GET BYTE TO AL FROM UART
; Input byte from UART to AL. To keep from confusing
; the send routine we must or in the present status to a
; local copy since we will clear the transmit ready status
; when we read.
; ALTERS: AL=character
;-------------------------------------------------------
_gbyt1: mov     al,siocs  ; get current status
        and     al,udr+utbre        ; mask all but RX, TX ready
        or      al,ustat  ; or in old status
        mov     ustat,al  ; and save
        and     al,udr              ; is byte available
        jz      _gbyt1              ; no, loop till
        and     ustat,utbre         ; reset ready
        mov     al,siodat           ; get char in al
        ret                         ; and return
getbyt  endp
;
;=======================================================
getword proc    near    ;get a word into bx register (2 bytes from
UART)
; ALTERS: AL=bl+bh (no carry), BX=word (received high-byte first)
;-------------------------------------------------------
                call    getbyt
                mov             bh, al
                call    getbyt
                mov             bl, al
                add             al, bh
                ret
getword endp
;
;=======================================================
```

```
delay   proc    near    ; DELAY 50 MILLISECONDS
; Delay for appoximatly 50 milliseconds based on 4 MHz
; 8088.
; ALTERS: nothing
;-------------------------------------------------------
        push    ax                  ; save reg
        mov     ax,11111            ; init counter
_dly:   dec     ax                  ;  2 ; 4.5 uS / loop
        jnz     _dly                ; 16 ;  @ 4.0 MHz
        pop     ax                  ; restore
        ret                         ; and return
delay   endp
;
;=======================================================
delay2  proc    near    ; DELAY 5 MILLISECONDS
; Delay for appoximatly 5 milliseconds based on 4 MHz
; 8088.
; ALTERS: nothing
;-------------------------------------------------------
        push    ax                  ; save reg
        mov     ax,1111             ; init counter
_dly2:  dec     ax                  ;  2 ; 4.5 uS / loop
        jnz     _dly2               ; 16 ;  @ 4.0 MHz
        pop     ax                  ; restore
        ret                         ; and return
delay2  endp
;
;
;=======================================================
bin2dec proc    near    ;CONVERT BINARY TO ASCII DECIMAL
;Alters: AX, DI, DX      BX is input
;-------------------------------------------------------
        push    cx
        push    si
        mov     di, offset _dec_v  ;start of curr val
        mov     byte ptr [di], '0'    ;set result to zero
        mov     byte ptr [di+1], '0'
        mov     byte ptr [di+2], '0'
        mov     byte ptr [di+3], '0'
        mov     byte ptr [di+4], '0'
        mov     dx, 15              ;15 significant bits
        mov     si, offset ascdat  ;table of ascii bit vals
        rol     bx, 1               ;check sign
        jnc     poswd
        not     bx                  ;negate
        inc     bx                  ;and form 2's complement of
```

```
        inc     bx                      ;rotated value
        mov     byte ptr [di+5], '-' ;sign
        jmp     short b2a               ;done with sign
poswd:  mov     byte ptr [di+5], '+'            ;sign
b2a:    rol     bx, 1                   ;look at top bit
        jc      doadd                   ;if bit is 1
        add     si, 5                   ;else next power of 2
        jmp     nadd                    ;and pass add loop
doadd:  mov     di, offset _dec_v ;start of curr val
        mov     cx, 5                   ;5 dec digits for 16 bit
        mov     ah, 0                   ;no carry at start
ascadd: mov     al, cs:[si]             ;get bit val digit (in rom)
        shr     ah, 1                   ;check for prev carry, clear
        adc     al, [di]    ;add current val digit
        aaa                             ;ascii adjust
        or      al, 30h                 ;convert to digit char
        mov     [di], al    ;store current
        inc     si                      ;inc pointers
        inc     di
        loop    ascadd                  ;go for next digit if CX>0
nadd:   dec     dx                      ;check bit count
        jnz     b2a                     ;until all 16
        mov     di, offset _dec_v[4]            ;start of curr val
        pop     si
        pop     cx
        ret
bin2dec endp
;
ascdat  db      '48361','29180','69040','84020','42010'
        db      '21500','65200','82100','46000','23000'
        db      '61000','80000','40000','20000','10000'
;
;
```

RX_INC.DS

```
;RX_INC.DS
;INCLUDE FILE FOR 3DAPS RECEIVER CODE (RX1A,RX3)
;after RAM definitions in data segment
;=======================================================
; ---- 81C55 Parallel I/O,RAM,Timer ----
;=======================================================
;
rx_inc segment at onboard
;
        org     pio+400h
;
piocsr  db      ?               ; pio command/status reg.
piopa   db      ?               ; port A
piopb   db      ?               ; port B
piopc   db      ?               ; port C
piotcl  db      ?               ; timer count lsb
piotcm  db      ?               ; timer ms & mode
;
; COMMAND REGISTER
painp   equ     0b              ; port a = input
paoutp  equ     1b              ; port a = output
pbinp   equ     0b * 2          ; port b = input
pboutp  equ     1b * 2          ; port b = output
pcinp   equ     00b * 4         ; port c = input
pcoutp  equ     11b * 4         ; port c = output
pcsta   equ     01b * 4         ; port c = output & A strobe
pcstab  equ     10b * 4         ; port c = strobed A & B
paei    equ     1b * 16         ; enable port a interrupt
padi    equ     0b * 16         ; disable port a interrupt
pbei    equ     1b * 32         ; enable port b interrupt
pbdi    equ     0b * 32         ; disable port b interrupt
tstop   equ     01b * 64        ; stop timer
tstopc  equ     10b * 64        ; stop timer at term. count
tstart  equ     11b * 64        ; start timer
;
; STATUS REGISTER
tc      equ     01000000b       ; terminal count flag
pbief   equ     00100000b       ; port b interrupt enabled
pbbfl   equ     00010000b       ; port b buffer full/empty
pbirq   equ     00001000b       ; port b interrupt request
paief   equ     00000100b       ; port a interrupt enabled
pabfl   equ     00000010b       ; port a buffer full/empty
pairq   equ     00000001b       ; port a interrupt request
;
```

```
; TIMER MODE
tssqwav  equ     00b * 64        ; single square wave
tcsqwav  equ     01b * 64        ; continuous square wave
tspulse  equ     10b * 64        ; ONE pulse on term count
tcpulse  equ     11b * 64        ; continuous pulses
;
;=================================================
; ---- 82C52 Serial Controller Interface ----
;=================================================
;
         org     sio
siodat   db      ?               ; UART data register
;
         org     sio+100h
siocs    db      ?               ; UART control and status
;
         org     sio+200h
siomcr   db      ?               ; modem control register
;
         org     sio+300h
siobrs   db      ?               ; bit rate select register
;
; CONTROL REGISTER
onestop  equ     0b              ; one stop bit
twostop  equ     1b              ; two stop bits
evenpar  equ     000b * 2        ; even parity
oddpar   equ     010b * 2        ; odd parity
nopar    equ     110b * 2        ; no  parity
fiveb    equ     00b * 16        ; five bit word length
sevenb   equ     10b * 16        ; seven bit word length
eightb   equ     11b * 16        ; eight bit word length
;
; BIT RATE REGISTER (note we use 1/10 normal crystal freq.)
b19200   equ     10000010b       ; div 2
b9600    equ     10000110b       ; div 4
b4800    equ     10001110b       ; div 8
b2400    equ     10010110b       ; div 16
b1200    equ     10100010b       ; div 64
b300     equ     10101010b       ; div 128
;
; MODEM CONTROL REGISTER
rtshigh  equ     0b              ; set request to send high
rtslow   equ     1b              ; set request to send low
dtrhigh  equ     0b * 2          ; set data term. ready high
dtrlow   equ     1b * 2          ; set data term. ready low
sioei    equ     1b * 4          ; enable rupts  (data sheet)
siodi    equ     0b * 4          ; disable rupts (**wrong**)
rcvren   equ     1b * 32         ; enable receiver
mdmei    equ     1b * 64         ; rupt on CTS,DSR change
mdmdi    equ     0b * 64         ; disable " "
;
; UART STATUS REGISTER
udr      equ     10000000b       ; uart receive data ready
utbre    equ     01000000b       ; transmit buffer ready
utc      equ     00100000b       ; transmission complete
ums      equ     00010000b       ; uart modem status change
urbrk    equ     00001000b       ; break received
uoe      equ     00000100b       ; overrun error
ufe      equ     00000010b       ; frame error
upe      equ     00000001b       ; parity error
;
; MODEM STATUS REGISTER
cts      equ     01b             ; clear to send
dsr      equ     10b             ; data set ready
;
;=================================================
; ---- 58274 Real Time Clock ----
;=================================================
;
         org     rtc
;
rtccsr   db      ?       ; command and status register
tenths   db      ?       ; tenths of seconds
seconds  db      ?       ;
tensecs  db      ?       ;
minutes  db      ?       ;
tenmins  db      ?       ;
hours    db      ?       ;
tenhour  db      ?       ;
days     db      ?       ;
tendays  db      ?       ;
months   db      ?       ;
tenmons          db      ?       ;
years    db      ?       ;
tenyear  db      ?       ;
dayweek          db      ?       ;
rtcmode  db      ?       ; clock setting & rupt register
;
rx_inc ends
```

RX_INC.SQR:

```
;=============================================================
sqrt_30 proc    near
;-------------------------------------------------------
                push    di
                push    si
                push    bx
                push    cx
                mov     di, dx
                mov     si, ax
                mov     bx, 4000h       ;top bit
                mov     ax, bx
                imul    bx              ;Xt is in DX:AX
                cmp     dx, di
                je      tlo14
                ja      ep14
                jb      en14
tlo14:          cmp     ax, si
                jne     je14
                jmp     done    ;long je
je14:           ja      ep14
en14:           add     bx, 2000h
                jmp     end14
ep14:           sub     bx, 2000h
end14:          mov     ax, bx
                imul    bx              ;Xt is in DX:AX
                cmp     dx, di
                je      tlo13
                ja      ep13
                jb      en13
tlo13:          cmp     ax, si
                jne     je13
                jmp     done    ;long je
je13:           ja      ep13
en13:           add     bx, 1000h
                jmp     end13
ep13:           sub     bx, 1000h
end13:          mov     ax, bx
                imul    bx              ;Xt is in DX:AX
                cmp     dx, di
                je      tlo12
                ja      ep12
                jb      en12
tlo12:          cmp     ax, si
                jne     je12
                jmp     done    ;long je
je12:           ja      ep12
en12:           add     bx, 0800h
                jmp     end12
ep12:           sub     bx, 0800h
end12:          mov     ax, bx
                imul    bx              ;Xt is in DX:AX
                cmp     dx, di
                je      tlo11
                ja      ep11
                jb      en11
tlo11:          cmp     ax, si
                jne     je11
                jmp     done    ;long je
je11:           ja      ep11
en11:           add     bx, 0400h
                jmp     end11
ep11:           sub     bx, 0400h
end11:          mov     ax, bx
                imul    bx              ;Xt is in DX:AX
                cmp     dx, di
                je      tlo10
                ja      ep10
                jb      en10
tlo10:          cmp     ax, si
                jne     je10
                jmp     done    ;long je
je10:           ja      ep10
en10:           add     bx, 0200h
                jmp     end10
ep10:           sub     bx, 0200h
end10:          mov     ax, bx
                imul    bx              ;Xt is in DX:AX
                cmp     dx, di
                je      tlo9
                ja      ep9
                jb      en9
tlo9:           cmp     ax, si
                jne     je9
                jmp     done    ;long je
je9:            ja      ep9
en9:            add     bx, 0100h
                jmp     end9
ep9:            sub     bx, 0100h
end9:           mov     ax, bx
                imul    bx              ;Xt is in DX:AX
```

```
        cmp     dx, di
        je      tlo8
        ja      ep8
        jb      en8
tlo8:   cmp     ax, si
        jne     je8
        jmp     done    ;long je
je8:    ja      ep8
en8:    add     bx, 0080h
        jmp     end8
ep8:    sub     bx, 0080h
end8:   mov     ax, bx
        imul    bx              ;Xt is in DX:AX
        cmp     dx, di
        je      tlo7
        ja      ep7
        jb      en7
tlo7:   cmp     ax, si
        jne     je7
        jmp     done    ;long je
je7:    ja      ep7
en7:    add     bx, 0040h
        jmp     end7
ep7:    sub     bx, 0040h
end7:   mov     ax, bx
        imul    bx              ;Xt is in DX:AX
        cmp     dx, di
        je      tlo6
        ja      ep6
        jb      en6
tlo6:   cmp     ax, si
        jne     je6
        jmp     done    ;long je
je6:    ja      ep6
en6:    add     bx, 0020h
        jmp     end6
ep6:    sub     bx, 0020h
end6:   mov     ax, bx
        imul    bx              ;Xt is in DX:AX
        cmp     dx, di
        je      tlo5
        ja      ep5
        jb      en5
tlo5:   cmp     ax, si
        jne     je5
        jmp     done    ;long je
je5:    ja      ep5
en5:    add     bx, 0010h
        jmp     end5
ep5:    sub     bx, 0010h
end5:   mov     ax, bx
        imul    bx              ;Xt is in DX:AX
        cmp     dx, di
        je      tlo4
        ja      ep4
        jb      en4
tlo4:   cmp     ax, si
        jne     je4
        jmp     done    ;long je
je4:    ja      ep4
en4:    add     bx, 0008h
        jmp     end4
ep4:    sub     bx, 0008h
end4:   mov     ax, bx
        imul    bx              ;Xt is in DX:AX
        cmp     dx, di
        je      tlo3
        ja      ep3
        jb      en3
tlo3:   cmp     ax, si
        jne     je3
        jmp     done    ;long je
je3:    ja      ep3
en3:    add     bx, 0004h
        jmp     end3
ep3:    sub     bx, 0004h
end3:   mov     ax, bx
        imul    bx              ;Xt is in DX:AX
        cmp     dx, di
        je      tlo2
        ja      ep2
        jb      en2
tlo2:   cmp     ax, si
        jne     je2
        jmp     done    ;long je
je2:    ja      ep2
en2:    add     bx, 0002h
        jmp     end2
ep2:    sub     bx, 0002h
end2:   mov     ax, bx
        imul    bx              ;Xt is in DX:AX
        cmp     dx, di
```

```
                je      tlo1
                ja      ep1
                jb      en1
tlo1:           cmp     ax, si
                je      done
                ja      ep1
en1:            add     bx, 0001h
                jmp     end1
ep1:            sub     bx, 0001h
end1:           mov     ax, bx
                imul    bx              ;check - is it closest?
                mov     temp4, ax
                mov     temp4[2], dx
                sub     temp4, si
                sbb     temp4[2], di
                cmp     temp4[2], 0     ;take abs value
                jnl     pos_e
                neg     temp4
                not     temp4[2]
pos_e:          cmp     dx, di
                je      tlo0
                ja      ep0
                jb      en0
tlo0:           cmp     ax, si
                je      done
                ja      ep0
en0:            mov     cx, 1
                jmp     end0
ep0:            mov     cx, -1
end0:           add     bx, cx
                mov     ax, bx
                imul    bx              ;Xt is in DX:AX
                sub     ax, si
                sbb     dx, di
                cmp     dx, 0           ;take abs value
                jnl     cmp_e
                neg     ax
                not     dx
cmp_e:          cmp     dx, temp4[2]
                je      tloA
                jb      done
                ja      enA
tloA:           cmp     ax, temp4
                jbe     done
enA:            sub     bx, cx
done:           mov     ax, bx

                pop     cx
                pop     bx
                pop     si
                pop     di
                ret                     ;=== end of sqrt_30
sqrt_30         endp
;
;
^Z
```

M3DAPSA.C:

```
#include <stdio.h>
#include <math.h>
#include <gf.h>
#include <timedate.h>
#include <asiports.h>
#include <conio.h>
#include <color.h>
#define RXBUFLEN 100
#define TXBUFLEN 100
static unsigned char input_file[20], output_file[20], xceive[65], xmit[25],
            crange[8][4][2], initialxmit[55], lowbyte, highbyte, line[80];
void onscreenr(), onscreeng(), onscreenc();
main()
{
/***********************************************/
/***********************************************/
/**********                           **********/
/**********       M3DAPSA.C           **********/
/**********                           **********/
/**********       Version 2.0         **********/
/**********                           **********/
/**********     Multipoint 3DAPS      **********/
/**********    Communications and     **********/
/**********    Monitoring Program     **********/
/**********                           **********/
/***********************************************/
/***********************************************/
/*        Begin Variable Declarations        */
   struct TIMEDATE *ptd, *sgettime();
/*                                   */
    char yn[3], wrong[5], c, plus, minus, ay, pe, pm;
    char hw[2], sosn[2], sosd[2], nth, xth[8][2], yth[8][2];
    char zth[8][2], xhy[4][2], yhy[4][2], zhy[4][2], inputk;
/*                                   */
    int stattxm3daps, statopencom, checksum, j, k, hwidth;
    int check, check1, range[8][4], port, startv, mark;
    int i, noth, sosnum, sosden, xthump[8], ythump[8], zthump[8];
    int xhyd[4], yhyd[4], zhyd[4], test, asiclear(), asiputb();
    int elim, lowest[4], save, sequence=0, stop_flag=0;
/*                                   */
    unsigned int m3dapstxbuffree, gettxfree(), m3dapsrxcount;
    unsigned int getrxcnt(), statrxm3daps;
    unsigned int ulbyte, uhbyte;
/*                                   */
    float sos;
/*                                   */
    FILE *fp, *fopen(), *ifp, *ofp;
/*                                   */
    while (!stop_flag)
   {
      mark = startv = test = 0;
      save = 0;
      grad2 = derror = distance = 0.0;
/*************************************************/
/*****                                     *****/
/*****   Begin Variables Initialization    *****/
/*****                                     *****/
/*************************************************/
      for (i=0; i<8; ++i)
     {
        xthump[i] = ythump[i] = zthump[i] = 0;
     }
     cls();
      printf("\nDo you wish to:");
      printf("\n      (a)  input the data");
      printf("\n      (b)  use a data file");
      printf("\n      ----> :");
      scanf("%s", yn);
      if (yn[0] == 'b')
     {
/*************************************************/
/*****                                     *****/
/*****    Input data from file             *****/
/*****                                     *****/
/*************************************************/
          printf("\nInput the data file name :");
          scanf("%s", input_file);
          ifp = fopen(input_file, "r");
          fscanf(ifp, "%d", &hwidth);
          fscanf(ifp, "%d", &sosnum);
          fscanf(ifp, "%d", &sosden);
          fscanf(ifp, "%d", &noth);
          for (i=0; i<noth; ++i)
        {
            fscanf(ifp, "%d%d%d", &xthump[i], &ythump[i],
                        &zthump[i]);
        }
         for (i=0; i<4; ++i)
         {
            fscanf(ifp, "%d%d%d", &xhyd[i], &yhyd[i], &zhyd[i]);
```

```
        }
        fclose(ifp);
        goto dspl;
    }
/*****************************************************/
/*****                                       *****/
/*****      Input data from keyboard         *****/
/*****                                       *****/
/*****************************************************/
hwdt:   printf("Input the Gate Half-Width : ");
    scanf("%d", &hwidth);
    if (test == 1)
    {
        goto dspl;
    }
sosn:   printf("\nInput the Speed-Of-Sound Numerator : ");
    scanf("%d", &sosnum);
    if (test == 1)
    {
        goto dspl;
    }
sosd:   printf("\nInput the Speed-Of-Sound Denominator : ");
    scanf("%d", &sosden);
    if (test == 1)
    {
        goto dspl;
    }
not:    printf("\nInput the Number Of Thumpers : ");
    scanf("%d", &noth);
xyzt:   for (i=0; i<noth; ++i)
    {
        printf("\nInput X Y Z for Thumper #%1d : ", i);
        scanf("%d%d%d", &xthump[i], &ythump[i], &zthump[i]);
    }
    if (test == 1)
    {
        goto dspl;
    }
xyzh:   for (i=0; i<4; ++i)
    {
        printf("\nInput X Y Z for Hydrophone #%1d : ", i);
        scanf("%5d%5d%5d", &xhyd[i], &yhyd[i], &zhyd[i]);
    }
dspl:   printf("\n");
    sos = (float) sosnum / (float) sosden * 2000.;
    printf("Gate Half-Width is %5d\n", hwidth);
    printf("Speed of Sound is %6.2f\n", sos);
    printf("Thumper Positions (mm)\n");
    printf("Number   X      Y      Z");
    for (i=0; i<noth; ++i)
    {
printf("\n  %d    %d    %d    %d",i,xthump[i],ythump[i],zthump[i]);
    }
    printf("\nHydrophone Positions (mm)\n");
    printf("Number   X      Y      Z");
    for (i=0; i<4; ++i)
    {
printf("\n  %d    %d    %d    %d", i, xhyd[i], yhyd[i], zhyd[i]);
    }
    test = 0;
    printf("\nIs data correct? ");
    scanf("%s", yn);
    if (yn[0] == 'n')
    {
        test = 1;
        printf("\nWhich input is wrong:");
        printf("\nGate Half-Width (hwdt)");
        printf("\nSpeed-of-Sound Numerator (sosn)");
        printf("\nSpeed-of-Sound Denominator (sosd)");
        printf("\nNumber of Thumpers (noth)");
        printf("\nThumper Coordinates (xyzt)");
        printf("\nHydrophone Coordinates (xyzh)");
        printf("\nInput the 4-letter code : ");
        scanf("%s", wrong);
        if (wrong[0] == 'h')
        {
            goto hwdt;
        }
        if (wrong[3] == 'n')
        {
            goto sosn;
        }
        if (wrong[3] == 'd')
        {
            goto sosd;
        }
        if (wrong[0] == 'n')
        {
            goto not;
        }
        if (wrong[3] == 't')
        {
```

```
            goto xyzt;
        }
         if (wrong[3] == 'h')
        {
            goto xyzh;
        }
    }
      printf("\nDo you wish to save the initial data?");
     scanf("%s", yn);
     if (yn[0] == 'y')
     {
         printf("\n          Output data file name :");
          scanf("%s", output_file);
          ofp = fopen(output_file, "w");
          fprintf(ofp, "%5d\n", hwidth);
          fprintf(ofp, "%4d\n", sosnum);
          fprintf(ofp, "%4d\n", sosden);
          fprintf(ofp, "%4d\n", noth);
         for (i=0; i<noth; ++i)
        {
            fprintf(ofp, "%5d %5d %5d\n", xthump[i], ythump[i],
                        zthump[i]);
        }
         for (i=0; i<4; ++i)
        {
            fprintf(ofp, "%5d %5d %5d\n", xhyd[i], yhyd[i], zhyd[i]);
        }
         fclose(ofp);
    }
          printf("\n\nDo you wish to save the results? ");
          scanf("%s", yn);
          if (yn[0] == 'y')
    {
/***********************************************/
/***** Save Range Data in File          *****/
/***********************************************/
             save = 1;
              printf("\nType Output file name : ");
              scanf("%s", output_file);
              ofp = fopen(output_file, "w");
         gets(line);
          printf("\nInput a header for the data file. Three lines max");
          printf("\n\n");
         for (i=0; i<3; ++i)
        {
            gets(line);
            fprintf(ofp, "\n%s", line);
        }
          printf("\n");
          fprintf(ofp, "\n");
           }
        printf("\nIs this the first time this program has been run");
       printf("\n  since the receiver was turned on? ");
       scanf("%s", yn);
/***********************************************/
/*****                                 *****/
/*****       Initialization Complete       *****/
/*****       Begin Main Multipoint 3-DAPS  *****/
/*****       Communications Routines       *****/
/*****                                 *****/
/***********************************************/
      plus = '+';
      minus = '-';
      cls();
           printf("\nUhura, open a hailing frequency.....");
/***********************************************/
/*****   Open a COM channel to the receiver    *****/
/***********************************************/
      statopencom = asiopen(COM1, (ASINOUT|BINARY|NORMALRX),
RXBUFLEN,
                  TXBUFLEN, 9600, P_NONE, 1, 8, ON, ON);
      if (statopencom != ASSUCCESS)
      {
         printf("\nASIOPEN error %d     ", statopencom);
         stop_flag = 1;
      }
       if (stop_flag)
      {
         break;
      }
        printf(".....Channel Opened Successfully");
         printf("\nBegging Starfleet to Begin.....");
/***********************************************/
/*****   Send Receiver a + or a <CR>,          *****/
/*****    Depending on whether we just started *****/
/*****   Or are continuing                     *****/
/***********************************************/
      asiclear(COM1, ASIN);
      c = (yn[0] == 'y') ? 13 : plus;
/***********************************************/
/***** Send a <CR> if Starting New      *****/
/***** Send a '+' if continuing         *****/
```

```
/*********************************************/
      stattxm3daps = asiputc(COM1, c);
      if (_aserror != ASSUCCESS)
      {
          printf("\nError In Pleading %d", stattxm3daps);
         stop_flag = 1;
      }
       if (stop_flag)
      {
         break;
      }
       if (yn[0] == 'y')
      {
         for (i=0; i<100; ++i)
        {
/*********************************************************
/
/***** Wait for Receiver to Respond to Startup       *****/
/*********************************************************
/
              onscreenc(24,33,0,"Waiting...");
             m3dapsrxcount = getrxcnt(COM1);
             if (m3dapsrxcount == 1)
            {
                pm = asigetc(COM1);
               if (pm == '+')
               {
                  break;
               }
               else
               {
                   stop_flag = 1;
               }
            }
             if (i == 99)
             {
                 onscreenr(10,1,0,"No answer received. Check Receiver");
                stop_flag = 1;
             }
        }
      }
       if (stop_flag)
      {
         break;
      }
        printf("......Plea Accepted\n");
        cls();
/*****************************************************/
/*****    Clear the Transmit and Receive Buffers *****/
/*****************************************************/
          onscreenr(23,1,0,"Hit Carriage Return to Exit.");
        asiclear(COM1, ASIN);
        asiclear(COM1, ASOUT);
        ay = 'A';
        pe = 'P';
        c = ay;
/*****************************************************/
/*****    Send the Receiver an 'A' for init          *****/
/*****************************************************/
          onscreenc(1,1,0,"Transmitting character A.....");
         stattxm3daps = asiputc(COM1, c);
        if (_aserror != ASSUCCESS)
        {
            onscreenr(2,1,0,"\nError In A Xmit %d", stattxm3daps);
           stop_flag = 1;
        }
         if (stop_flag)
        {
           break;
        }
           onscreenc(1,31,0,".....Character transmitted");
/*****************************************************/
/*****    Begin loading the init data into       *****/
/*****    character variables for transference   *****/
/*****    into the Transmit buffer               *****/
/*****************************************************/
/*****    Protocol: High byte FIRST              *****/
/*****              Low  byte SECOND             *****/
/*****************************************************/
        hw[0] = hwidth / 256;
         hw[1] = hwidth - 256*(hwidth/256);
         sosn[0] = sosnum / 256;
          sosn[1] = sosnum - 256*(sosnum/256);
         sosd[0] = sosden / 256;
          sosd[1] = sosden - 256*(sosden/256);
        nth = noth;
        for (i=0; i<8; ++i)
       {
           xth[i][0] = xthump[i]/256;
            xth[i][1] = xthump[i] - 256*(xthump[i]/256);
           yth[i][0] = ythump[i]/256;
            yth[i][1] = ythump[i] - 256*(ythump[i]/256);
```

```
        zth[i][0] = zthump[i]/256;
        zth[i][1] = zthump[i] - 256*(zthump[i]/256);
      }
       initialxmit[0] = hw[0];
       initialxmit[1] = hw[1];
       initialxmit[2] = sosn[0];
       initialxmit[3] = sosn[1];
       initialxmit[4] = sosd[0];
       initialxmit[5] = sosd[1];
       initialxmit[6] = nth;
      for (i=8; i<56;)
      {
         j = (i-8)/6;
          initialxmit[i-1] = xth[j][0];
          initialxmit[i] = xth[j][1];
          initialxmit[i+1] = yth[j][0];
          initialxmit[i+2] = yth[j][1];
          initialxmit[i+3] = zth[j][0];
          initialxmit[i+4] = zth[j][1];
         i += 6;
      }
/*********************************************************/
/*****                                  *****/
/*****    Initialization Complete. Begin Download  *****/
/*****                                  *****/
/*********************************************************/
senda:     onscreenc(2,1,0,"Begin Init Download.....");
         stattxm3daps = asiputb(COM1, initialxmit, 55);
       if (_aserror != ASSUCCESS)
       {
           onscreenr(3,1,0,"Error Transmitting M3DAPS Buffer %d",
stattxm3daps);
          stop_flag = 1;
       }
        if (stop_flag)
       {
          break;
       }
         onscreenc(2,26,0,".....End Init Download");
        m3dapstxbuffree = gettxfree(COM1);
         onscreenc(3,1,0,"Waiting on Empty TX Buffer.....");
        while ((m3dapstxbuffree < 55) && (!stop_flag))
       {
          m3dapstxbuffree = gettxfree(COM1);
          if ( kbhit())
          {
              stop_flag = 1;
          }
       }
        if (stop_flag)
       {
          break;
       }
         onscreenc(3,33,0,".....TX Buffer Empty");
/*********************************************************/
/******  Repeat Init Download                    *****/
/*********************************************************/
          onscreenc(4,1,0,"Begin Repeat Init Download.....");
         stattxm3daps = asiputb(COM1, initialxmit, 55);
       if (_aserror != ASSUCCESS)
       {
           onscreenr(5,1,0,"Error Transmitting Repeat M3DAPS Buffer %d",
               stattxm3daps);
          stop_flag = 1;
       }
        if (stop_flag)
       {
          break;
       }
         onscreenc(4,33,0,".....End Repeat Init Download");
/*********************************************************/
/*                                  */
/*     Waiting for Receiver to Reply          */
/*                                  */
/*********************************************************/
        m3dapsrxcount = getrxcnt(COM1);
         onscreenc(5,1,0,"Waiting on Acknowledge.....");
        while ((m3dapsrxcount != 1) && (!stop_flag))
       {
              m3dapsrxcount = getrxcnt(COM1);
          if ( kbhit())
          {
              stop_flag = 1;
          }
       }
        if (stop_flag)
       {
          break;
       }
         onscreenc(5,29,0,".....Acknowledge Received");
         onscreenc(6,1,0,"Begin Acknowledge Decoding.....");
       pm = asigetc(COM1);
```

```
        onscreenc(6,33,0,".....End Acknowledge Decoding >>> %1c", pm);
      if (pm == minus)
      {
         goto senda;
      }
       for (i=0; i<4; ++i)
      {
           onscreenc(15+i,2,0,"%d",i);
      }
       for (j=0; j<noth; ++j)
      {
           onscreenc(14,8*j+7,0,"%d",j);
      }
/*                                      */
/*     Multipoint Receiver has successfully acquired      */
/*     the Initialization data                            */
/*                                      */
            ptd = sgettime(4);
      if (save == 1)
      {
           fprintf(ofp, "\nStart Time: %s", ptd->dateline);
      }
       while (!stop_flag)
      {
/***
****
****    Begin Hydrophone Coordinate Download
****
***/
          c = pe;
             onscreenc(1,1,0,"Transmitting character P.....");
           stattxm3daps = asiputc(COM1, c);
          if (_aserror != ASSUCCESS)
          {
               onscreenr(2,1,0,"Error In P Xmit %d", stattxm3daps);
              stop_flag = 1;
          }
           if (stop_flag)
          {
             break;
          }
             onscreenc(1,31,0,".....Character transmitted");
sendp:       for (i=0; i<4; ++i)
          {
/***
****
```

```
****    Convert Hydrophone Positions to download String
****
***/
            xhy[i][0] = xhyd[i] / 256;
             xhy[i][1] = xhyd[i] - 256*(xhyd[i]/256);
            yhy[i][0] = yhyd[i] / 256;
             yhy[i][1] = yhyd[i] - 256*(yhyd[i]/256);
            zhy[i][0] = zhyd[i] / 256;
             zhy[i][1] = zhyd[i] - 256*(zhyd[i]/256);
        }
         for (i=1; i<25;)
        {
            j = (i-1)/6;
             xmit[i-1] = xhy[j][0];
             xmit[i] = xhy[j][1];
             xmit[i+1] = yhy[j][0];
             xmit[i+2] = yhy[j][1];
             xmit[i+3] = zhy[j][0];
             xmit[i+4] = zhy[j][1];
            i += 6;
        }
         checksum = 0;
         for (i=0; i<4; ++i)
         {
/***
****
****    Compute Checksum
****
***/
            checksum += (xhyd[i]/256) + (xhyd[i] - 256*(xhyd[i]/256)) +
                    (yhyd[i]/256) + (yhyd[i] - 256*(yhyd[i]/256)) +
                    (zhyd[i]/256) + (zhyd[i] - 256*(zhyd[i]/256));
        }
          xmit[24] = checksum - 256*(checksum/256);
            onscreenc(2,1,0,"Begin Hphone Download.....");
         asiclear(COM1, ASOUT);
/***
****
****    Transmit Data
****
***/
          stattxm3daps = asiputb(COM1, xmit, 25);
         if (_aserror != ASSUCCESS)
         {
              onscreenr(3,1,0,"Error in transmitting Hydrophone Data");
            stop_flag = 1;
```

```
        }
         if (stop_flag)
        {
            break;
        }
           onscreenc(2,27,0,".....End Hphone Download");
         m3dapsrxcount = getrxcnt(COM1);
           onscreenc(3,1,0,"Waiting on Acknowledge.....");
         while (m3dapsrxcount != 1)
        {
/***
****
****   Wait for Receiver to Acknowledge
****
***/
            m3dapsrxcount = getrxcnt(COM1);
        }
           onscreenc(3,29,0,".....Acknowledge Received");
           onscreenc(4,1,0,"Begin Hphone Acknowledge.....");
        pm = asigetc(COM1);
          onscreenc(4,30,0,".....End Hphone Acknowledge >>> %1c", pm);
        if (pm == minus)
        {
            goto sendp;
        }
/*                                  */
/*    Multipoint Receiver has successfully acquired      */
/*    the Hydrophone coordinates                  */
/*                                  */
getit:      onscreenc(5,1,0,"Waiting On Empty RX Buffer.....");
         m3dapsrxcount = getrxcnt(COM1);
          while ((m3dapsrxcount != 8*noth+1) && (!stop_flag))
        {
/***
****
****   Wait for Input Buffer to fill with full set of Ranges
****
***/
            m3dapsrxcount = getrxcnt(COM1);
           if ( kbhit())
          {
              inputk = getch();
              if (inputk == 'm')
             {
/***
****
****   'm' input Marks Data
****
***/
                    ++mark;
                     onscreenr(11,50,0,"Mark #%3d", mark);
                    if (save == 1)
                    {
                        fprintf(ofp, "\nMark #%3d", mark);
                    }
                }
/***
****
****   Carriage-Return input stops run
****
***/
               if (inputk == 13)
              {
                  stop_flag = 1;
              }
            }
          }
           if (stop_flag)
          {
              break;
          }
           ptd = sgettime(4);
            onscreenc(5,33,0,".....RX Buffer Full   ");
             onscreenc(6,1,0,"Begin Range Acquisition.....");
/***
****
****   Pull in Ranges from buffer
****
***/
             statrxm3daps = asigetb(COM1, xceive, 8*noth+1);
           if (_aserror != ASSUCCESS)
          {
              onscreenr(7,30,0,"Error in Receiving Ranges");
             stop_flag = 1;
          }
           if (stop_flag)
          {
              break;
          }
            onscreenc(6,30,0,".....End Range Reception            ");
/***
****
```

```
****   Convert Ranges from Char data to Int
****
***/
        for (i=0; i<8*noth;)
        {
            k = i/8;
            for (j=0; j<4; ++j)
            {
                crange[k][j][0] = xceive[i+2*j];
                crange[k][j][1] = xceive[i+2*j+1];
            }
            i += 8;
        }
        check = 0;
        for (i=0; i<noth; ++i)
        {
            for (j=0; j<4; ++j)
            {
/***
****
****   Calculate checksum of data
****
***/
                lowbyte = crange[i][j][1];
                highbyte = crange[i][j][0];
                ulbyte = lowbyte;
                uhbyte = highbyte;
                range[i][j] = uhbyte*256 + ulbyte;
                check += crange[i][j][0] + crange[i][j][1];
            }
        }
        pm = check;
        if (pm != xceive[8*noth])
        {
/***
****
****   If checksums don't match, get ranges again
****
***/
            pm = minus;
            asiclear(COM1, ASIN);
            stattxm3daps = asiputc(COM1, pm);
            onscreenr(7,1,0,"Sending Down Checksum >>> %c ", pm);
            goto getit;
        }
        pm = plus;
        onscreenc(7,1,0,"Sending Down Checksum >>> %c ", pm);
        stattxm3daps = asiputc(COM1, pm);
/*                                          */
/*    We now have the ranges from each thumper to each      */
/*    hydrophone                            */
/*                                          */
/*                                          */
/*                                          */
        for (i=0; i<4; ++i)
        {
            if (save == 1)
            {
                fprintf(ofp, "\n %d ", i);
            }
            for (j=0; j<noth; ++j)
            {
/***
****
****   Display Ranges to User
****
***/
                onscreenc(15+i,8*j+5,0,"%d   ", range[j][i]);
                if (save == 1)
                {
                    fprintf(ofp, " %d ", range[j][i]);
                }
            }
        }
        if (save == 1)
        {
            fprintf(ofp, "\n");
        }
        ++sequence;
        onscreenr(22,35,0,"Sequence #%d", sequence);
        onscreenc(23,1,0,"Waiting Next Sequence.....");
        asiclear(COM1, ASIN);
        asiclear(COM1, ASOUT);
    }
    if (stop_flag)
    {
        break;
    }
}
endgame:  onscreenr(22,1,0,"Exiting Main Program. Bye-Bye.");
          onscreenr(23,1,0,"Termination Complete.");
          fclose(ofp);
```

```
      exit(0);
}
void  onscreenc(y,x,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9)
    int y,x,page;
   char *fmt;
     unsigned  a0,a1,a2,a3,a4,a5,a6,a7,a8,a9;
{
       curset(y,x,page);
          rprintf(CYAN,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9);
       return;
}
void  onscreenr(y,x,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9)
    int y,x,page;
   char *fmt;
     unsigned  a0,a1,a2,a3,a4,a5,a6,a7,a8,a9;
{
       curset(y,x,page);
          rprintf(LTRED,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9);
       return;
}
void  onscreeng(y,x,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9)
    int y,x,page;
   char *fmt;
     unsigned  a0,a1,a2,a3,a4,a5,a6,a7,a8,a9;
{
       curset(y,x,page);
          rprintf(WHITE,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9);
       return;
}
```

M3FREADE.C:

```
#include <stdio.h>
#include <math.h>
#include <gf.h>
#include <timedate.h>
#include <asiports.h>
#include <conio.h>
#include <color.h>
#define RADDEG 57.29577951
#define MARGIN 0.000001
#define RXBUFLEN 100
#define TXBUFLEN 100
static double itv[4][4][4], error2, margin2;
static char init[20], init_file[30], output_file[30], mof[20],
          line80[80], input_file[30], file_name[20], line[10],
          time[10], date[10], mat_out_file[30], mark1[6], mark2[5],
          mark3[5], output_f[20], *data = "data\\", *output = "output\\";
static int range[8][4];
void onscreeng(), onscreenc(), onscreenr(), onscreenw(), onscreeny();
main()
{
/*********************************************************
***/
/*********************************************************
***/
/******                                          ******/
/******              M3FREADE.C                  ******/
/******                                          ******/
/******               Version 1.0                ******/
/******                                          ******/
/******     Multipoint 3DAPS Calculation Software    ******/
/******                                          ******/
/*********************************************************
***/
/*********************************************************
***/
/*        Begin Variable Declarations          */
      struct TIMEDATE *ptd, *sgettime();
/*******************************************************/
       char wrong[5], plus, minus, pe, pm, c, yn[3];
       char nth, *strcpy();
/*******************************************************/
       int j, k, hwidth, m, n, o, mark, strlen(), encode;
        int i, noth, sosnum, sosden, xthump[8], ythump[8], zthump[8];
        int xhyd[4], yhyd[4], zhyd[4], test, again=1, liter;
```

```
        int elim, save, kiter, angle, endfl, dumb, lowest[4];
/*********************************************************/
        float distance, derror, cost[4], refcost[4], deltax;
        float deltay, deltaz, refcx[4], refcy[4], refcz[4], refcx2[4];
        float refcy2[4], refcz2[4], grad2[4], lowcost[4], costx[4];
        float costy[4], costz[4], costx2[4], costy2[4], costz2[4];
        float fcost[4], error[4], sos, xm[4], ym[4], zm[4];
        float refx[4], refy[4], refz[4], x[4], y[4], z[4];
        float xf[4], yf[4], zf[4], correct_range;
        float xx, xy, xz, yx, yy, yz, zx, zy, zz;
        float nx, ny, nz, ox, oy, oz, theta[4], phi[4], psi[4];
        float bix, cix, ciy, k1, k2, b1, b2, c1, c2;
       float ax, ay, az;
/*********************************************************/
        FILE *fp, *fopen(), *ifp, *ofp, *mofp;
/*********************************************************/
        long encoder, old_encoder, delta_e;
/*********************************************************/
    while (again == 1)
    {
/***
****
****  Initialize Variables and Filenames
****
****
***/
        strcpy(init_file, data);
        strcpy(input_file, data);
        strcpy(output_file, output);
        strcpy(mat_out_file, data);
       encode = encoder = old_encoder = delta_e = 0;
       save = 0;
       angle = 0;
       mark = 0;
       derror = distance = 0.0;
       for (i=0; i<4; ++i) {
           costx[i] = costy[i] = costz[i] = 0.0;
           costx2[i] = costy2[i] = costz2[i] = 0.0;
           grad2[i] = lowcost[i] = fcost[i] = 0.0;
          refcx[i] = refcy[i] = refcz[i] = 0.0;
          refcx2[i] = refcy2[i] = refcz2[i] = 0.0;
          refcost[i] = cost[i] = 0.0;
          refx[i] = refy[i] = refz[i] = 0.0;
          xf[i] = yf[i] = zf[i] = 0.0;
          xm[i] = ym[i] = zm[i] = 0.0;
          x[i]  = y[i]  = z[i]  = 0.0;
          theta[i] = 0.0;
          phi[i] = 0.0;
          psi[i] = 0.0;
          lowest[i] = 77;
          error[i] = 0.0;
       }
/***
****
****  Request Range Data file from user
****
***/
        printf("\nThis Software assumes all data is located in");
       printf("\n          C:\\KILLER\\DATA");
        printf("\nAnd all output will be located in");
       printf("\n          C:\\KILLER\\OUTPUT");
        printf("\n\n");
        printf("\nName of Ranges data file?  ");
        scanf("%s", file_name);
         strcpy(&input_file[5], file_name);
         printf("\n\nLooking for %s", input_file);
       if (strcmp(input_file, "stdout") == 0)
       {
          fp = stdout;
       }
       else
       {
           fp = fopen(input_file, "r");
       }
        printf("\nInitialization: (F)ile or (K)eyboard -- ");
/***
****
****  Request Initialization Data
****
***/
       scanf("%s", yn);
       if ((yn[0] == 'F') || (yn[0] == 'f'))
       {
           printf("\nName of Initialization data file? ");
          scanf("%s", init);
           strcpy(&init_file[5], init);
           printf("\n\nLooking for %s", init_file);
          ifp = fopen(init_file, "r");
          fscanf(ifp, "%d", &hwidth);
          fscanf(ifp, "%d", &sosnum);
          fscanf(ifp, "%d", &sosden);
          fscanf(ifp, "%d", &noth);
```

```
        for (i=0; i<noth; ++i) {
            fscanf(ifp, "%d%d%d", &xthump[i], &ythump[i],
                        &zthump[i]);
        }
        for (i=0; i<4; ++i) {
            fscanf(ifp, "%d%d%d", &xhyd[i], &yhyd[i], &zhyd[i]);
        }
        fclose(ifp);
        goto dspl;
    }
    test = 0;
not:   printf("\nInput the Number Of Thumpers : ");
    scanf("%d", &noth);
xyzt:   for (i=0; i<noth; ++i)
    {
        printf("\nInput X Y Z for Thumper #%1d : ", i);
        scanf("%d%d%d", &xthump[i], &ythump[i], &zthump[i]);
    }
    if (test == 1)
    {
        goto dspl;
    }
xyzh:   for (i=0; i<4; ++i)
    {
        printf("\nInput X Y Z for Hydrophone #%1d : ", i);
        scanf("%d%d%d", &xhyd[i], &yhyd[i], &zhyd[i]);
    }
dspl:   printf("\n");
    printf("Thumper Positions (mm)\n");
    printf("Number   X      Y      Z");
    for (i=0; i<noth; ++i)
    {
    printf("\n %d    %6d    %6d    %6d",i,xthump[i],ythump[i],zthump[i]);
    }
    printf("\nHydrophone Positions (mm)\n");
    printf("Number   X      Y      Z");
    for (i=0; i<4; ++i)
    {
    printf("\n %d    %6d    %6d    %6d", i, xhyd[i], yhyd[i], zhyd[i]);
    }
    test = 0;
    printf("\nIs data correct? ");
    scanf("%s", yn);
    if (yn[0] == 'n')
    {
        test = 1;
        printf("\nWhich input is wrong:");
        printf("\nNumber of Thumpers (noth)");
        printf("\nThumper Coordinates (xyzt)");
        printf("\nHydrophone Coordinates (xyzh)");
        printf("\nInput the 4-letter code : ");
        scanf("%s", wrong);
        if (wrong[0] == 'n')
        {
            goto not;
        }
        if (wrong[3] == 't')
        {
            goto xyzt;
        }
        if (wrong[3] == 'h')
        {
            goto xyzh;
        }
    }
    printf("\nDo you wish to save the initialization data? ");
    scanf("%s", yn);
    if (yn[0] == 'y')
    {
        printf("\n     Name of Init File : ");
        scanf("%s", output_f);
        ofp = fopen(output_f, "w");
        fprintf(ofp, "%d %d %d %d\n", hwidth, sosnum, sosden, noth);
        for (i=0; i<noth; ++i)
        {
            fprintf(ofp, "%d %d %d\n", xthump[i], ythump[i],
                zthump[i]);
        }
        for (i=0; i<4; ++i)
        {
            fprintf(ofp, "%d %d %d\n", xhyd[i], yhyd[i], zhyd[i]);
        }
        fclose(ofp);
    }
/*********************************************************
***/
/*                                                       */
/*      Initialization Complete                          */
/*      Begin Main Multipoint 3-DAPS Routines            */
/*                                                       */
/*********************************************************
***/
```

```
      printf("\nDo you wish to save the results? ");
      scanf("%s", yn);
      if (yn[0] == 'y')
      {
         save = 1;
/***
****
****  Save Coordinates
****
***/
          strcpy(&output_file[7], &input_file[5]);
          for (i=0; i<strlen(output_file); ++i)
         {
             if (output_file[i] == '.')
            {
                output_file[i+1] = 'o';
                output_file[i+2] = 'u';
                output_file[i+3] = 't';
               break;
            }
         }
          printf("\n\n Ouptut File Name = %s", output_file);
          ofp = fopen(output_file, "w");
      }
      printf("\nNumber of iterations : ");
/***
****
****  Set Iteration Max
****
***/
      kiter = 0;
      liter = 1;
      scanf("%d", &kiter);
      if (kiter == 0)
      {
         goto endgame;
      }
/*********************************************************
***/
/****                                ****/
/****       Request Angle Calculation Engage         ****/
/****                                ****/
/*********************************************************
***/
      printf("\nDo you wish to calculate angles? ");
      scanf("%s", yn);
      if (yn[0] == 'y')
      {
/***
****
****  Calculate Angles
****
***/
         angle = 1;
         margin2 = MARGIN*MARGIN;
          printf("\nDo you wish to input iTv from the keyboard or a file?");
         printf("\n          (k) or (f) ---> ");
         scanf("%s", yn);
         if (yn[0] == 'k') {
/***
****
****  Input iTv Matrices from keyboard
****
***/
            printf("\nInput iTv values from keyboard:");
           for (n=0; n<4; ++n) {
              for (m=0; m<4; ++m) {
                 for (o=0; o<4; ++o) {
                    printf("\n     iTv[%d][%d][%d] = ", m, o, n);
                     scanf("%lf", &itv[m][o][n]);
                 }
              }
           }
         }
         else
         {
/***
****
****  Input iTv Matrices from File
****
***/
            printf("\nInput Filename of matrix data file :");
           scanf("%s", mof);
            strcpy(&mat_out_file[5], mof);
             printf("\n\nLooking for %s", mat_out_file);
            mofp = fopen(mat_out_file, "r");
           for (n=0; n<4; ++n)
           {
              for (m=0; m<4; ++m)
              {
                 for (o=0; o<4; ++o)
                 {
```

```
                    fscanf(mofp, "%lf", &itv[m][o][n]);
                }
            }
        }
        fclose(mofp);
    }
dspla:  for (n=0; n<4; ++n) {
        printf("\n");
        for (m=0; m<4; ++m) {
            printf("\n");
            for (o=0; o<4; ++o) {
                printf(" %11.10f ", itv[m][o][n]);
            }
        }
    }
    printf("\nAre these values correct?");
    scanf("%s", yn);
    if (yn[0] == 'y') {
        goto sav;
    }
    printf("\nWhich value is wrong?");
    scanf("%d%d%d", &m, &o, &n);
    printf("\nInput iTv[%d][%d][%d] :", m, o, n);
    scanf("%lf", &itv[m][o][n]);
    goto dspla;
sav:    printf("\nDo you wish to save the matrix data? ");
    scanf("%s", yn);
    if (yn[0] == 'y') {
        printf("\nInput Matrix save file name : ");
        scanf("%s", mof);
        strcpy(&mat_out_file[5], mof);
        mofp = fopen(mat_out_file, "w");
        for (n=0; n<4; ++n) {
            for (m=0; m<4; ++m) {
                for (o=0; o<4; ++o) {
                    fprintf(mofp, " %11.10f ", itv[m][o][n]);
                }
                fprintf(mofp, "\n");
            }
            fprintf(mofp, "\n");
        }
        fclose(mofp);
    }
}
cls();
if (angle == 1)
{
 onscreeng(16,1,0,"Hydrophone    Theta    Phi    Psi");
}
 onscreenw(9,10,0,"Iteration : ");
 onscreenw(9,30,0,"Sequence : ");
 onscreenr(10,1,0,
   "Hydrophone   X    Y    Z    ERROR    ELIM");
for (i=0; i<4; ++i)
{
/***
****
**** Strip off first 4 lines from file - Header and extra newline
****
***/
    fgets(line80, 80, fp);
    if (save == 1)
    {
        fprintf(ofp, "%s", line80);
    }
    onscreenc(i,1,0,"%s", line80);
}
fgets(line80, 80, fp);
/***
****
**** Determine if file contains encoder values too
****
***/
if (line80[0] == 'E')
{
    encode = 1;
    onscreenc(4,1,0,"%s", line80);
    fgets(line80, 80, fp);
}
if (save == 1)
{
    fprintf(ofp, "%s\n", line80);
}
skippy:  endfl = 0;
fscanf(fp, "%s", line);
if (save == 1)
{
    fprintf(ofp, "%s", line);
}
if (line[0] == 'M')
{
/***
```

```
****
**** Check for Marks in data set
****
***/
        mark = 1;
        for (i=0; i<5; ++i)
        {
            mark1[i] = line[i];
        }
        for (i=0; i<2; ++i)
        {
            fscanf(fp, "%s", line);
            if (i == 0)
            {
                for (j=0; j<5; ++j)
                {
                    mark2[j] = line[j];
                }
            }
            if (i == 1)
            {
                for (j=0; j<5; ++j)
                {
                    mark3[j] = line[j];
                }
            }
            if (save == 1)
            {
                fprintf(ofp, " %s", line);
            }
        }
        if (save == 1)
        {
            fprintf(ofp, "\n");
        }
        fscanf(fp, "%s", line);
        if (save == 1)
        {
            fprintf(ofp, "%s", line);
        }
    }
/***
****
**** Get Time is... line from file
****
***/
        for (i=1; i<5; ++i)
        {
            fscanf(fp, "%s", line);
            if (i == 2)
            {
                for (j=0; j<10; ++j)
                {
                    date[j] = line[j];
                }
            }
            if (i == 4)
            {
                for (j=0; j<10; ++j)
                {
                    time[j] = line[j];
                }
            }
            if (save == 1)
            {
                fprintf(ofp, " %s", line);
                if (i == 4)
                {
                    fprintf(ofp, "\n");
                }
            }
        }
        for (j=0; j<4; ++j)
        {
/***
****
**** Input range values from file. If at end, exit
****
***/
            if ((fscanf(fp, "%d", &dumb)) == EOF)
            {
                endfl = 1;
                break;
            }
            for (i=0; i<noth; ++i)
            {
                fscanf(fp, "%d", &range[i][j]);
            }
        }
        if (encode != 0)
        {
            old_encoder = encoder;
```

```
            fscanf(fp, "%ld", &encoder);
        }
        if (endfl == 1)
            goto endgame;
/*********************************************************
********/
/*                                        */
/*  This line above here is what I added!! (Matt)          */
/*                                        */
/*      We now have the ranges from each thumper to each    */
/*     hydrophone                         */
/*                                        */
/*********************************************************
********/
        for (i=0; i<4; ++i) {
            lowcost[i] = 7000000000.0;
            refx[i] = (float)xhyd[i];
            refy[i] = (float)yhyd[i];
            refz[i] = (float)zhyd[i];
            xf[i] = (float)xhyd[i];
            yf[i] = (float)yhyd[i];
            zf[i] = (float)zhyd[i];
        }
/***
****
****  Begin First Coordinate Calculations with all valid range values
****
***/
        for (i=0; i<4; ++i) {
            k=0;
thump1:  /*  printf("Ref.(%d) XYZ= %5d %5d
%5d\n",i,refx[i],refy[i],refz[i]);*/
            refcost[i] = refcx[i] = refcy[i] = refcz[i] = 0.0;
            refcx2[i] = refcy2[i] = refcz2[i] = 0.0;
            for (j=0; j<noth; ++j) {
                if (range[j][i] == 0) {
                    continue;
                }
                deltax = refx[i] - (float)xthump[j];
                deltay = refy[i] - (float)ythump[j];
                deltaz = refz[i] - (float)zthump[j];
/***
****
****  Calculate distance from est. hydrophone posn to thumpers
****
***/
                distance = sqrt((deltax*deltax + deltay*
                        deltay + deltaz*deltaz));
                derror = distance - range[j][i];
/***
****
****  Calculate error in estimated range.
****  Calculate Derivatives for Newton-Raphson
***/
                refcost[i] += derror*derror;
                refcx[i] += deltax*derror/distance;
                refcy[i] += deltay*derror/distance;
                refcz[i] += deltaz*derror/distance;
                refcx2[i] += deltax*deltax*correct_range/(
                  distance*distance*distance) + 1.0 - correct_range/distance;
                refcy2[i] += deltay*deltay*correct_range/(
                  distance*distance*distance) + 1.0 - correct_range/distance;
                refcz2[i] += deltaz*deltaz*correct_range/(
                  distance*distance*distance) + 1.0 - correct_range/distance;
            }
            ++k;
/***
****
****  Compute Gradient
****
***/
            grad2[i] = refcx[i]*refcx[i] + refcy[i]*refcy[i] + refcz[i]*
                  refcz[i];
            if (grad2[i]>8.0 && k != 10) {
/***
****
****  If high enough, change estimated coords.
****
***/
                refx[i] += -0.5*refcx[i]/refcx2[i];
                refy[i] += -0.5*refcy[i]/refcy2[i];
                refz[i] += -0.5*refcz[i]/refcz2[i];
                goto thump1;
            }
        }
/***
****
****  Begin set of 8 loops with (max) noth-1 valid ranges
****
***/
        for (elim=0; elim<noth; ++elim) {
            for (i=0; i<4; ++i) {
```

```
            x[i] = (float)xhyd[i];
            y[i] = (float)yhyd[i];
            z[i] = (float)zhyd[i];
         }
/***
****
****  Calculations are the same as before; at least one range is
****  eliminated from each set
****
***/
         for (i=0; i<4; ++i) {
            k=0;
thump2: /*        printf("XYZ (%d, elim=%d) = %5d %5d %5d\n",i,elim,
              x[i],y[i],z[i]);*/
             cost[i] = costx[i] = costy[i] = costz[i] = 0.0;
             costx2[i] = costy2[i] = costz2[i] = 0.0;
             for (j=0; j<noth; ++j) {
                if ((j == elim) || (range[j][i] == 0)) {
                   continue;
                }
                deltax = x[i] - (float)xthump[j];
                deltay = y[i] - (float)ythump[j];
                deltaz = z[i] - (float)zthump[j];
                distance = sqrt((deltax*deltax
                          + deltay*deltay + deltaz*deltaz));
                 correct_range = ((float)range[j][i] - 111.72)/1.0066;
                derror = distance - correct_range;
                cost[i] += derror*derror;
                costx[i] += deltax*derror/distance;
                costy[i] += deltay*derror/distance;
                costz[i] += deltaz*derror/distance;
                costx2[i] += deltax*deltax*correct_range/(
              distance*distance*distance) + 1.0 - correct_range/distance;
                 costy2[i] += deltay*deltay*correct_range/(
              distance*distance*distance) + 1.0 - correct_range/distance;
                 costz2[i] += deltaz*deltaz*correct_range/(
              distance*distance*distance) + 1.0 - correct_range/distance;
             }
             ++k;
              grad2[i] = costx[i]*costx[i] + costy[i]*costy[i] +
                    costz[i]*costz[i];
             if (grad2[i]>8.0 && k != 10) {
                x[i] += -0.5*costx[i]/costx2[i];
                y[i] += -0.5*costy[i]/costy2[i];
                z[i] += -0.5*costz[i]/costz2[i];
                goto thump2;
             }
/***
****
****  Find set of coords with lowest "cost"
****
***/
             if (cost[i] < lowcost[i]) {
                lowest[i] = elim;
                lowcost[i] = cost[i];
                xf[i] = x[i];
                yf[i] = y[i];
                zf[i] = z[i];
             }
          }
       }
/***
****
****  If "cost" of above estimate is less than half of full-set
****  "cost" use above estimate.
****
***/
       for (i=0; i<4; ++i) {
          if (lowcost[i] < 0.5*refcost[i]) {
             fcost[i] = lowcost[i];
          }
          else {
             lowest[i] = 99;
             fcost[i] = refcost[i];
             xf[i] = refx[i];
             yf[i] = refy[i];
             zf[i] = refz[i];
          }
          error[i] = sqrt(fcost[i]);
          xm[i] = xf[i]/1000.;
          ym[i] = yf[i]/1000.;
          zm[i] = zf[i]/1000.;
       }
       ptd = sgettime(4);
        onscreenc(6,11,0,"Pool Time is %s - %s", date, time);
        onscreenw(7,10,0,"Local Time is %s", ptd->dateline);
       if (mark == 1)
       {
            onscreenr(5,10,0,"%s %s %s", mark1, mark2, mark3);
          mark = 0;
       }
       else
```

```
    {
        onscreenr(5,9,0,"                ");
    }
/***
****
****  Display values on the screen
****
***/
     for (i=0; i<4; ++i)
     {
         onscreenc(11+i,5,0,"%d",i);
         onscreenc(11+i,14,0,"%6.4f      ",xm[i]);
         onscreenc(11+i,22,0,"%6.4f      ",ym[i]);
         onscreenc(11+i,30,0,"%6.4f      ",zm[i]);
         onscreenr(11+i,39,0,"%6.4f      ",error[i]);
         onscreeny(11+i,51,0," %2d  ",lowest[i]);
       if (save == 1)
       {
           fprintf(ofp,
           "\n %d  %6.4f  %6.4f  %6.4f  %6.4f  %2d",
             i,  xm[i],  ym[i],  zm[i], error[i], lowest[i]);
       }
       if (encode != 0)
       {
           delta_e = encoder - old_encoder;
           onscreeng(8,11,0,"%ld       ", encoder);
           onscreeng(8,30,0,"%ld       ",old_encoder);
           onscreeng(8,51,0,"%ld       ",delta_e);
           onscreeny(8,1,0,"Encoder =");
           onscreeny(8,21,0,"Old_E =");
           onscreeny(8,41,0,"Delta_E =");
          if ((save == 1) && (i == 3))
          {
            fprintf(ofp, "\n posn=%ld : delta=%ld", encoder, delta_e);
          }
       }
        xhyd[i] = (int)xf[i];
        yhyd[i] = (int)yf[i];
        zhyd[i] = (int)zf[i];
     }
      onscreenr(9,22,0," %d  ", kiter);
      onscreeng(9,42,0,"%d", liter);
     if (angle == 1) {
/*********************************************************
***/
```

```
/*********************************************************
***/
/*****                                  *****/
/*****       Begin Multipoint 3DAPS Attitudes       *****/
/*****           Determination              *****/
/*****                                  *****/
/*********************************************************
***/
/*********************************************************
***/
/***
****
****  Use only three hydrophones: four sets
****  (1) 0,1,2   (2) 1,2,3   (3) 2,3,0   (4) 3,0,1
****
****  Perform Coordinate Transformation to find Attitudes
****
***/
        for (n=0; n<4; ++n) {
          m = n + 1;
          if (m == 4) {
             m = 0;
          }
          o = m + 1;
          if (o == 4) {
             o = 0;
          }
           k1 = xf[m] - xf[n];
           k2 = xf[o] - xf[n];
           b1 = yf[m] - yf[n];
           b2 = yf[o] - yf[n];
           c1 = zf[m] - zf[n];
           c2 = zf[o] - zf[n];
            bix = sqrt(k1*k1 + b1*b1 + c1*c1);
            cix = (k1*k2 + b1*b2 + c1*c2)/bix;
            ciy = sqrt(k2*k2 + b2*b2 + c2*c2 - cix*cix);
           xx = k1/bix;
           xy = b1/bix;
           xz = c1/bix;
            yx = (k2 - (cix/bix)*k1)/ciy;
            yy = (b2 - (cix/bix)*b1)/ciy;
            yz = (c2 - (cix/bix)*c1)/ciy;
           zx =  xy*yz - yy*xz;
           zy = -xx*yz + yx*xz;
           zz =  xx*yy - yx*xy;
             nx = xx*itv[0][0][n] + yx*itv[1][0][n] + zx*itv[2][0][n] +
```

```
            xf[n]*itv[3][0][n];
          ny = xy*itv[0][0][n] + yy*itv[1][0][n] + zy*itv[2][0][n] +
            yf[n]*itv[3][0][n];
          nz = xz*itv[0][0][n] + yz*itv[1][0][n] + zz*itv[2][0][n] +
            zf[n]*itv[3][0][n];
          ox = xx*itv[0][1][n] + yx*itv[1][1][n] + zx*itv[2][1][n] +
            xf[n]*itv[3][1][n];
          oy = xy*itv[0][1][n] + yy*itv[1][1][n] + zy*itv[2][1][n] +
            yf[n]*itv[3][1][n];
          oz = xz*itv[0][1][n] + yz*itv[1][1][n] + zz*itv[2][1][n] +
            zf[n]*itv[3][1][n];
          ax = xx*itv[0][2][n] + yx*itv[1][2][n] + zx*itv[2][2][n] +
            xf[n]*itv[3][2][n];
          ay = xy*itv[0][2][n] + yy*itv[1][2][n] + zy*itv[2][2][n] +
            yf[n]*itv[3][2][n];
          az = xz*itv[0][2][n] + yz*itv[1][2][n] + zz*itv[2][2][n] +
            zf[n]*itv[3][2][n];
           theta[n] = atan2(-nz, nx*cos(psi[n])+ny*sin(psi[n]));
         error2 = (abs(nz) - 1.0)*(abs(nz) - 1.0);
         if (error2 < margin2) {
              phi[n] = atan2(-ax*sin(psi[n]) + ay*cos(psi[n]),
                        ox*sin(psi[n]) - oy*cos(psi[n]));
             onscreenr(17+n,55,0,"e2");
        }
        else {
            phi[n] = atan2(oz, az);
            psi[n] = atan2(ny, nx);
        }
    }
     for (n=0; n<4; ++n) {
          onscreenc(17+n,4,0," %d",n);
          onscreenc(17+n,15,0,"   %8.6f   ", theta[n]*RADDEG);
          onscreenc(17+n,29,0,"   %8.6f   ", phi[n]*RADDEG);
          onscreenc(17+n,42,0,"   %8.6f   ", psi[n]*RADDEG);
         if (save == 1) {
             fprintf(ofp, "\n %d   %8.6f   %8.6f   %8.6f",
             n, theta[n]*RADDEG, phi[n]*RADDEG, psi[n]*RADDEG);
        }
    }
     if (save == 1) {
         fprintf(ofp, "\n");
    }
   }        onscreenr(22,1,0,"Waiting Next Sequence.....");
--kiter;
++liter;
if (kiter != 0) {
            if (save == 1) {
                fprintf(ofp, "\n");
            }
             goto skippy;
        }
endgame:
        if (save == 1)
        {
            fclose(ofp);
        }
         onscreenc(22,1,0,"Another Data Set?  ");
         scanf("%s", yn);
         if (yn[0] == 'n')
        {
            again = 0;
        }
    }
     onscreenr(23,1,0,"Exiting Main Program.");
     onscreenr(24,1,0,"Termination Complete.");
    exit(0);
}
void  onscreenc(y,x,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9)
     int y,x,page;
    char *fmt;
      unsigned a0,a1,a2,a3,a4,a5,a6,a7,a8,a9;
{
        curset(y,x,page);
          rprintf(CYAN,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9);
        return;
}
void  onscreenr(y,x,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9)
     int y,x,page;
    char *fmt;
      unsigned a0,a1,a2,a3,a4,a5,a6,a7,a8,a9;
{
        curset(y,x,page);
          rprintf(LTRED,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9);
        return;
}
void  onscreenw(y,x,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9)
     int y,x,page;
    char *fmt;
      unsigned a0,a1,a2,a3,a4,a5,a6,a7,a8,a9;
{
        curset(y,x,page);
          rprintf(WHITE,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9);
```

```
    return;
}
void  onscreeng(y,x,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9)
    int y,x,page;
   char *fmt;
    unsigned a0,a1,a2,a3,a4,a5,a6,a7,a8,a9;
{
     curset(y,x,page);
       rprintf(GREEN,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9);
     return;
}
void  onscreeny(y,x,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9)
    int y,x,page;
   char *fmt;
    unsigned a0,a1,a2,a3,a4,a5,a6,a7,a8,a9;
{
     curset(y,x,page);
       rprintf(YELLOW,page,fmt,a0,a1,a2,a3,a4,a5,a6,a7,a8,a9);
     return;
}
```

SIMULATR.C:

```
#define TDELAY              0.22
#define SDELAY              2
#define COUNTS_PER_METER        52210.0

#define TRUE                1
#define FALSE               0

#include <stdio.h>
#include <math.h>

static int range[8][4];
main()
{
/*************************************************/
/*************************************************/
/**********                             **********/
/**********       SIMULATR.C            **********/
/**********    3DAPS Simulator Software **********/
/**********                             **********/
/**********     Version 1.0   9/21/89   **********/
/**********                             **********/
/**********      Karl G. Kowalski       **********/
/**********                             **********/
/*************************************************/
/*************************************************/
    int i, j, k, noth, xt[8], yt[8], zt[8], xn[4], yn[4], zn[4], xf[4], yf[4],
      zf[4], hwidth, sosn, sosd, again, command;
/*************************************************/
   long e, efar, enear, elast;
/*************************************************/
    float velocity, evel, delta_x, delta_y, delta_z, factor;
   float x[4], y[4], z[4];
/*************************************************/
    char yesno[3], file_name[20], line[80];
/*************************************************/
   FILE *ofp, *ifp, *fopen();
/*************************************************/
/**********                     **********/
/**********     Initialize Variables **********/
/**********                     **********/
/*************************************************/
   printf("\nWelcome to SIMULATR, the 3DAPS Simulation Program.");
    printf("\nBrought to you by the Holy One Himself.");
   printf("\n Would you care to input a data file? ");
```

```
    scanf("%s", yesno);
    if (yesno[0] == 'y')
    {
/***
****
****  Input data from File
****
***/
        printf("\n File to Open: ");
        scanf("%s", file_name);
        ifp = fopen( file_name, "r");
        fscanf(ifp, "%d", &noth);
        for (i=0; i<noth; ++i)
        {
            fscanf(ifp, "%d%d%d", &xt[i], &yt[i], &zt[i]);
        }
        fscanf(ifp, "%ld", &enear);
        for (i=0; i<4; ++i)
        {
            fscanf(ifp, "%d%d%d", &xn[i], &yn[i], &zn[i]);
        }
        fscanf(ifp, "%ld", &efar);
        for (i=0; i<4; ++i)
        {
            fscanf(ifp, "%d%d%d", &xf[i], &yf[i], &zf[i]);
        }
        fclose(ifp);
    }
    else
    {
/***
****
****  Input data from Keyboard
****
***/
        printf("\n Input Number of Thumpers : ");
        scanf("%d", &noth);
        printf("\n");
        for (i=0; i<noth; ++i)
        {
            printf("\n     Input Thumper %d XYZ : ", i);
            scanf("%d%d%d", &xt[i], &yt[i], &zt[i] );
        }
        printf("\n       Input Efar : ");
        scanf("%ld", &efar );
        printf("\n Input Far Hydrophone Values ");
        for (i=0; i<4; ++i)
        {
            printf("\n     Input Hydrophone %d XYZ : ", i);
            scanf("%d%d%d", &xf[i], &yf[i], &zf[i]);
        }
        printf("\n       Input Enear: ");
        scanf("%ld", &enear);
        printf("\n Input Near Hydrophone Values ");
        for (i=0; i<4; ++i)
        {
            printf("\n     Input Hydrophone %d XYZ : ", i);
            scanf("%d%d%d", &xn[i], &yn[i], &zn[i]);
        }
    }
    printf("\n Verify Data ");
    printf("\n Number of Thumpers = %d", noth);
    printf("\n Thumper Coordinates: ");
    for (i=0; i<noth; ++i)
    {
        printf("\n Xt[%d] Yt[%d] Zt[%d] = %d %d %d",i,i,i,xt[i],yt[i],zt[i]);
    }
    printf("\n\n Hydrophone Far Coordinates (Ef = %ld)", efar);
    for (i=0; i<4; ++i)
    {
        printf("\n Xf[%d] Yf[%d] Zf[%d] = %d %d %d",i,i,i,xf[i],yf[i],zf[i]);
    }
    printf("\n\n Hydrophone Near Coordinates (En = %ld)", enear);
    for (i=0; i<4; ++i)
    {
        printf("\n Xn[%d] Yn[%d] Zn[%d] = %d %d %d",i,i,i,xn[i],yn[i],zn[i]);
    }
    printf("\n\n Use Data? ");
    scanf("%s", yesno);
    while (yesno[0] == 'n')
    {
        printf("\n Input Number of Thumpers : ");
        scanf("%d", &noth);
        printf("\n");
        for (i=0; i<noth; ++i)
        {
            printf("\n     Input Thumper %d XYZ : ", i);
            scanf("%d%d%d", &xt[i], &yt[i], &zt[i] );
        }
        printf("\n       Input Efar : ");
        scanf("%ld", &efar );
        printf("\n Input Far Hydrophone Values ");
```

```
      for (i=0; i<4; ++i)
      {
          printf("\n     Input Hydrophone %d XYZ : ", i);
          scanf("%d%d%d", &xf[i], &yf[i], &zf[i]);
      }
      printf("\n       Input Enear: ");
      scanf("%ld", &enear);
      printf("\n Input Near Hydrophone Values ");
      for (i=0; i<4; ++i)
      {
          printf("\n     Input Hydrophone %d XYZ : ", i);
          scanf("%d%d%d", &xn[i], &yn[i], &zn[i]);
      }
      printf("\n Verify Data ");
      printf("\n Number of Thumpers = %d", noth);
      printf("\n Thumper Coordinates: ");
      for (i=0; i<noth; ++i)
      {
          printf("\n Xt[%d] Yt[%d] Zt[%d] = %d %d %d",i,i,i,xt[i],yt[i],zt[i]);
      }
      printf("\n\n Hydrophone Far Coordinates (Ef = %ld)", efar);
      for (i=0; i<4; ++i)
      {
          printf("\n Xf[%d] Yf[%d] Zf[%d] = %d %d %d",i,i,i,xf[i],yf[i],zf[i]);
      }
      printf("\n\n Hydrophone Near Coordinates (En = %ld)", enear);
      for (i=0; i<4; ++i)
      {
          printf("\n Xn[%d] Yn[%d] Zn[%d] = %d %d
%d",i,i,i,xn[i],yn[i],zn[i]);
      }
      printf("\n Use Data? ");
      scanf("%s", yesno);
   }
   printf("\n Save Data? ");
   scanf("%s", yesno);
   if (yesno[0] == 'y')
   {
/***
****
****  Save results to file
****
***/
      printf("\n Output File Name : ");
      scanf("%s", file_name);
      ofp = fopen( file_name, "w");
      fprintf(ofp, "\n%d", noth);
      for (i=0; i<noth; ++i)
      {
          fprintf(ofp, "\n%d %d %d", xt[i], yt[i], zt[i]);
      }
      fprintf(ofp, "\n%ld", enear);
      for (i=0; i<4; ++i)
      {
          fprintf(ofp, "\n%d %d %d", xn[i], yn[i], zn[i] );
      }
      fprintf(ofp, "\n%ld", efar);
      for (i=0; i<4; ++i)
      {
          fprintf(ofp, "\n%d %d %d", xf[i], yf[i], zf[i] );
      }
      fclose(ofp);
   }
   printf("\n\n");
   again = TRUE;
/*****************************************************/
/*****************************************************/
/**********                                 **********/
/**********  Begin Variable Initialization  **********/
/**********                                 **********/
/*****************************************************/
/*****************************************************/
   while (again)
   {
      printf("\n Experiment Output File Name : ");
      scanf("%s", file_name);
      ofp = fopen( file_name, "w");
      gets(line);
      printf("\n Enter Experiment File Header. Three Lines.");
      printf("\n\n");
      for (i=0; i<3; ++i)
      {
          gets(line);
          fprintf(ofp, "\n%s", line);
      }
      printf("\n");
      fprintf(ofp, "\nENCODER");
      fprintf(ofp, "\n");
      printf("\n Input Simulation Velocity (meters/second) : ");
      scanf("%f", &velocity);
      evel = velocity*COUNTS_PER_METER;
      for (k=0; k<10; ++k)
```

```
    {
/***
****
****  Begin Range Data File format
****  Save 10 sets of static ranges
****  For beginning of test
****
****
***/
        fprintf(ofp, "\n Time is : NO VALID TIME");
        printf("\n");
        for (i=0; i<4; ++i)
        {
            fprintf(ofp, "\n %d ", i);
            printf("\n %d ", i);
            for (j=0; j<noth; ++j)
            {
                delta_x = (float)(xf[i] - xt[j]);
                delta_y = (float)(yf[i] - yt[j]);
                delta_z = (float)(zf[i] - zt[j]);
                range[j][i] = (int)(sqrt(delta_x*delta_x +
                                delta_y*delta_y +
                                delta_z*delta_z));
                fprintf(ofp, " %d ", range[j][i] );
                printf(" %d ", range[j][i] );
            }
        }
        fprintf(ofp, "\n %ld\n", efar);
        printf("\n %ld\n", efar);
    }
    command = FALSE;
/*******************************************************/
/*******************************************************/
/**********                                   **********/
/**********       Begin Simulation            **********/
/**********                                   **********/
/*******************************************************/
/*******************************************************/
    e = efar;
    while ((e < enear) && (!command))
    {
/***
****
****  If we've passed the near end, stop run.
****
***/
        if (e >= enear)
        {
            break;
        }
        for (j=0; j<noth; ++j)
        {
/***
****
****  Calculate Position of Cart
****
***/
            factor = (float)(e-efar)/(float)(enear-efar);
            for (i=0; i<4; ++i)
            {
/***
****
****  If user hits a key, exit
****
***/
                if (kbhit())
                {
                    command = TRUE;
                }
/***
****
****  Calculate coordinates of Hydrophones based on cart posn.
****
***/
                x[i] = factor*(float)(xn[i] - xf[i]) + (float)xf[i];
                y[i] = factor*(float)(yn[i] - yf[i]) + (float)yf[i];
                z[i] = factor*(float)(zn[i] - zf[i]) + (float)zf[i];
                delta_x = x[i] - xt[j];
                delta_y = y[i] - yt[j];
                delta_z = z[i] - zt[j];
/***
****
****  Determine and store ranges.
****
***/
                range[j][i] = (int)sqrt(delta_x*delta_x +
                        delta_y*delta_y + delta_z*delta_z);
            }
            e += TDELAY*evel;
            if ( e > enear )
            {
                e = enear;
```

```
            }
          }
          if (kbhit())
          {
             command = TRUE;
          }
          for (i=0; i<4; ++i)
          {
              fprintf(ofp, "\n %d ", i);
              printf("\n %d ", i);
              for (j=0; j<noth; ++j)
             {
                  fprintf(ofp, " %d ", range[j][i] );
                  printf(" %d ", range[j][i] );
             }
          }
           fprintf(ofp, "\n %ld\n", e);
           printf("\n %ld\n", e);
          e += SDELAY*(long)TDELAY*evel;
       }
       if (!command)
       {
/***
****
**** If exited correctly, store 10 more static tests at near end
****
***/
          for (k=0; k<10; ++k)
          {
             for (i=0; i<4; ++i)
             {
                 fprintf(ofp, "\n %d ", i);
                 printf("\n %d ", i);
                 for (j=0; j<noth; ++j)
                 {
                     delta_x = (float)(xn[i] - xt[j]);
                     delta_y = (float)(yn[i] - yt[j]);
                     delta_z = (float)(zn[i] - zt[j]);
                    range[j][i] = sqrt(delta_x*delta_x + delta_y*delta_y +
                              delta_z*delta_z);
                     fprintf(ofp, " %d ", range[j][i]);
                     printf(" %d ", range[j][i]);
                 }
             }
              fprintf(ofp, "\n %ld\n", enear);
              printf("\n %ld\n", enear);
            }
            fclose(ofp);
             printf("\n Do you wish another Simulation? ");
             scanf("%s", yesno);
          }
          if ((yesno[0] == 'n') || (command))
          {
             again = FALSE;
          }
       }
       printf("\n Terminating Main Program");
       if (command)
       {
          fclose(ofp);
       }
       exit(0);
}
```

6.3 Appendix C. Vehicle-Relative Hydrophone Coordinates

The tables in this appendix list the coordinates of the hydrophones in their respective vehicle coordinate systems.

X	Y	Z	Error
0	0	-786	2
-630	0	-425	2
315	545	-425	2
315	-545	-425	2

All measurements in mm.

Table C-1. Cluster-Relative Hydrophone Coordinates

X	Y	Z	Error
-396	1090	128	2
-962	394	1280	2
178	-308	1290	2
-1466	-458	102	2

All measurements in mm.

Table C-2. BAT-Relative Hydrophone Coordinates (ref. 11)

X	Y	Z	Error
0	0	-1430	5
0	-1430	0	5
-1440	0	0	5
0	1434	0	5

All measurements in mm.

Table C-3. MPOD-Relative Hydrophone Coordinates

6.4 Appendix D. Tabular Data of Static Tests

Hyd./Thumper	Acoustic Range	Std. Dev.	Measured Range
0/1	5962	21	5850
0/2	5771	30	5645
0/3	6782	24	6710
0/4	6847	86	6520
0/5	10298	10	10100
0/6	10094	28	10000
0/7	10835	12	10620
0/8	10679	39	10520
1/1	6482	22	6375
1/2	6130	30	6005
1/3	7225	24	7150
1/4	7142	87	6830
1/5	9823	10	9615
1/6	9499	28	9400
1/7	10375	12	10160
1/8	10103	39	9940
2/1	6335	22	6220
2/2	5972	30	5850
2/3	6295	24	6220
2/4	6162	87	5845
2/5	10904	10	10700
2/6	10605	28	10500
2/7	10922	15	10705
2/8	10664	39	10490
3/1	5504	22	5390
3/2	5071	30	4940
3/3	7221	24	7155
3/4	7139	87	6830
3/5	10429	9	10230
3/6	10117	28	10010
3/7	11495	12	11260
3/8	11226	39	11050

All measurements in mm.

Table D-1. Static Test Range Results - MIT Pool

Hydrophone Coord.	Calculated Value	Std. Dev.	Measured Value
x_0	5331	40	5415
y_0	3701	24	3655
z_0	2016	79	2050
error	121	43	
x_1	5342	39	5420
y_1	4332	26	4285
z_1	2380	73	2460
error	121	36	
x_2	5880	38	5965
y_2	3370	24	3320
z_2	2379	83	2430
error	112	37	
x_3	4754	45	4840
y_3	3383	26	3330
z_3	2408	71	2480
error	117	37	

All measurements in mm.

Table D-2. Static Test Coordinate Results - MIT Pool

Hyd./Thumper	Acoustic Range	Std. Dev.	Measured Range
0/1	9664	52	9600
0/2	13899	29	13780
0/3	13756	21	13690
0/4	15055	24	14850
0/5	10271	38	10190
0/6	9108	32	9030
0/7	11255	16	11190
0/8	11914	2	11570
1/1	9203	52	9150
1/2	14064	27	13960
1/3	14156	21	14090
1/4	15795	23	15590
1/5	10025	39	9950
1/6	9052	32	8970
1/7	11671	16	11590
1/8	12450	2	12410
2/1	9740	52	9670
2/2	13577	82	13440
2/3	13115	21	13040
2/4	14901	24	14700
2/5	10925	39	10840
2/6	9694	32	9610
2/7	11735	16	11720
2/8	11977	2	11940
3/1	10269	52	10200
3/2	14550	93	14300
3/3	14009	21	13950
3/4	14973	24	14770
3/5	10529	39	10450
3/6	8597	33	8520
3/7	10803	16	10740
3/8	11356	2	11320

All measurements in mm.

Table D-3. Static Test Range Results - Marshall Tank

Hydrophone Coord.	Calculated Value	Std. Dev.
x_0	5810	32
y_0	-30	33
z_0	5241	62
error	303	44
x_1	5275	30
y_1	-362	28
z_1	4913	50
error	287	43
x_2	6342	48
y_2	-330	37
z_2	4868	73
error	281	53
x_3	5783	34
y_3	583	31
z_3	4922	48
error	287	48

All measurements in mm.

Hydrophone coordinates not measured directly.

Table D-4. Static Test Coordinate Results - Marshall Tank

6.5 Appendix E. IBM Encoder Circuit Diagram

Figures E-1 and E-2 show the circuit designed to interface between the encoder used in the controlled dynamic experiment (section 3.2) and the IBM PC/AT used to read and record range meaasurements. A modified version of the IBM software listed in Appendix B was used to perform both range data acquisition and encoder value data.

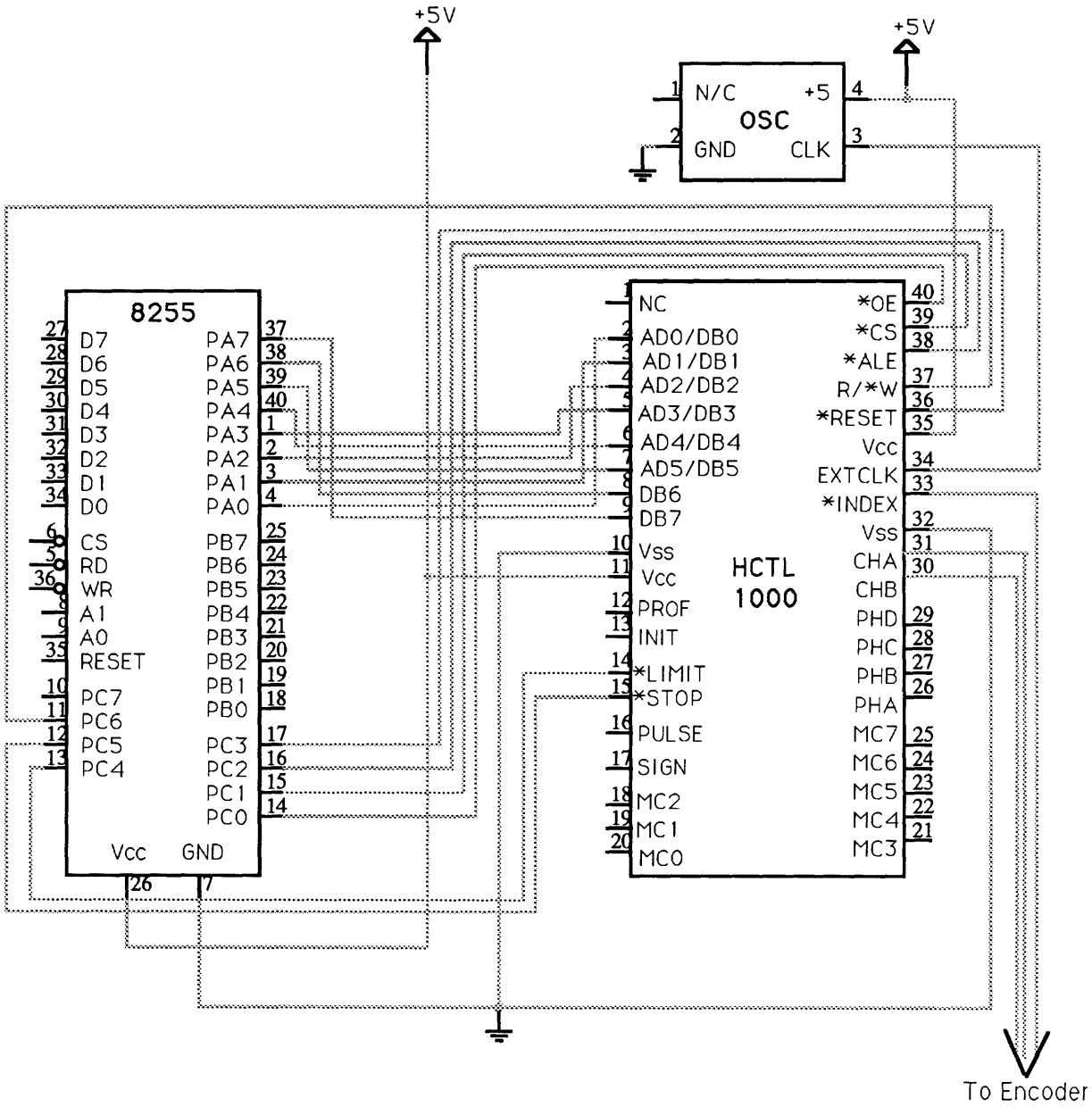


Figure E-1. Encoder Interface Circuit

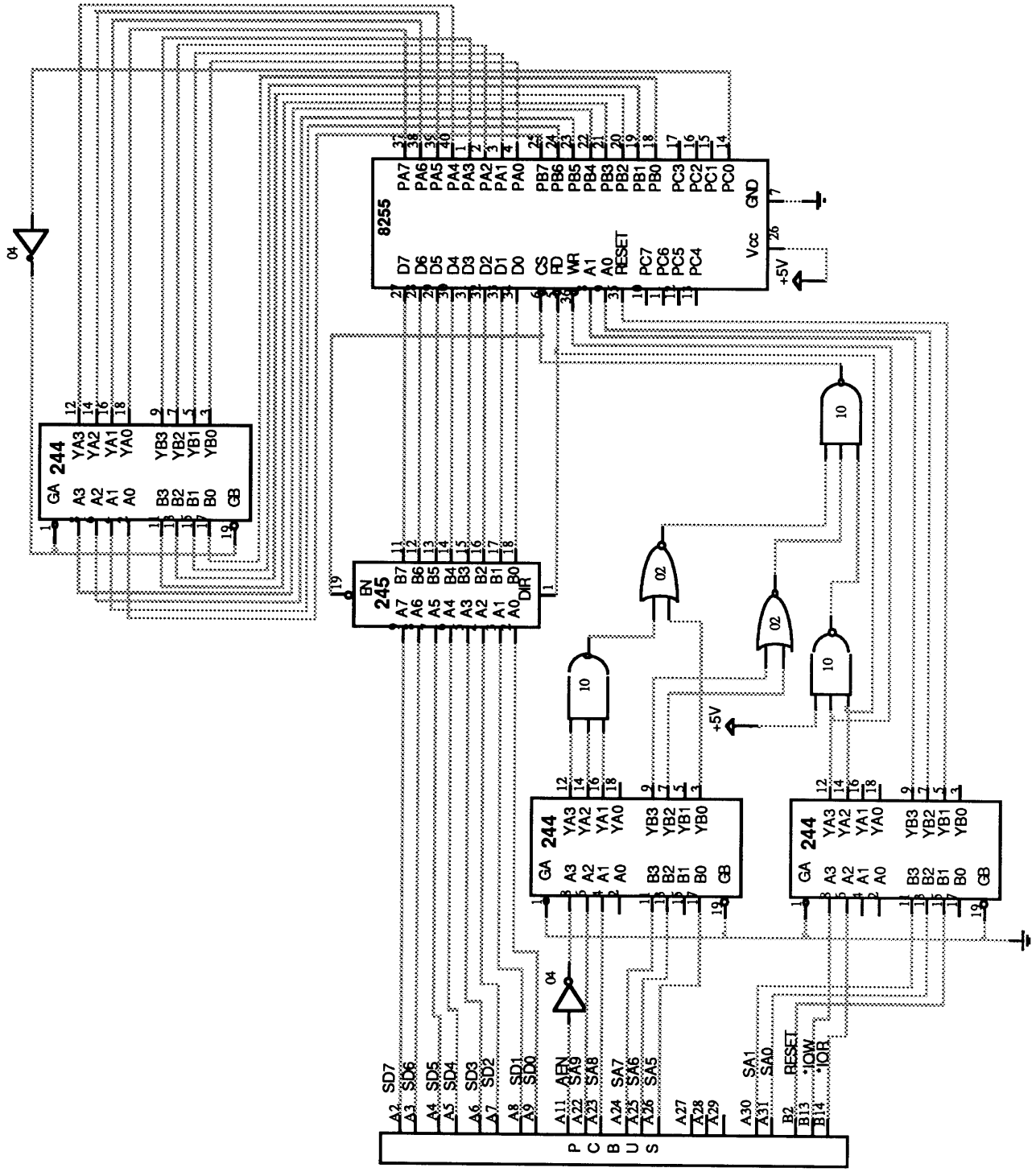


Figure E-2. Encoder Circuit IBM PC Bus Interface